CrossTalk

Nov / Dec 2011 **The Journal of Defense Software Engineering** Vol. 24 No. 6

Publisher's Choice

Report Documentation Page	*Form Approved* *OMB No. 0704-0188*

1. REPORT DATE **NOV 2011**	2. REPORT TYPE	3. DATES COVERED **00-11-2011 to 00-12-2011**
4. TITLE AND SUBTITLE **CrossTalk: The Journal of Defense Software Engineering. Volume 24, Number 6. November/December 2011**		5a. CONTRACT NUMBER
		5b. GRANT NUMBER
		5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)		5d. PROJECT NUMBER
		5e. TASK NUMBER
		5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **517 SMXS MXDEA,6022 Fir Ave,Hill AFB,UT,84056-5820**		8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)		10. SPONSOR/MONITOR'S ACRONYM(S)
		11. SPONSOR/MONITOR'S REPORT NUMBER(S)
12. DISTRIBUTION/AVAILABILITY STATEMENT **Approved for public release; distribution unlimited**		
13. SUPPLEMENTARY NOTES		
14. ABSTRACT		
15. SUBJECT TERMS		

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**	**Same as Report (SAR)**	**40**	

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std Z39-18

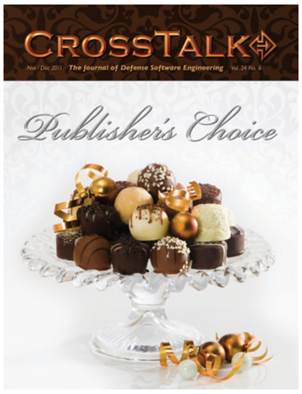
CONTENTS

Departments

3 From the Publisher

38 Upcoming Events

39 BackTalk


CrossTalk
Nov/Dec 2011 The Journal of Defense Software Engineering Vol. 24 No. 6
Publisher's Choice

Cover Design by
Kent Bingham

Publisher's Choice

4 **Netcentric Proxies for On-Orbit Sensors**
Proxies can provide support for security, policy enforcement, reliability, mediation, power, performance, and operational management. Proxies can also support information assurance by providing a means to enforce the separation of system components based on security policy and practices.
by Craig A. Lee and Samuel D. Gasster

8 **Give the Stakeholders What They Want: Design Peer Reviews the ATAM Style**
Looking at the results of many past ATAM evaluations, it becomes apparent that some mechanism is needed to better guide the architecture design process. Many ATAM evaluations show major issues with the system that could have been avoided with the right design approach.
by Felix Bachmann

11 **Software Architecture: Theory and Practice**
There is often a gap between widely accepted software engineering theory and practice. This is also true for the concept of software architecture. While the concept of software architecture has been in existence for quite some time, there is still a great deal of confusion over just what software architecture actually is.
by Michael Tarullo

16 **Utilizing Design of Experiments to Reduce IT System Testing Cost**
Utilizing Orthogonal Arrays (OA) for system and software testing will significantly reduce cost, schedule and risk. For the aerospace and defense industries, OA testing will help address the current environment of tighter budgets and schedules while ensuring end users promised performance.
by Madhav S. Phadke and Kedar M. Phadke

22 **Defect Management Using Depth of Inspection and the Inspection Performance Metric**
Advancement in fundamental engineering aspects of software development enables IT enterprises to develop a more cost effective and better quality product through aptly organized defect management strategies.
by T.R. Gopalakrishnan Nair and Suma. V.

28 **Free and Open Source Software Use: Benefits and Compliance Obligations**
Many systems developed for and deployed by the U.S. government now use Free and Open Source Software (FOSS). But FOSS use comes with potential license obligations. Essential compliance activities include identification of FOSS used in products along with communication of a FOSS bill of materials; review and approval of planned FOSS use; and satisfaction of license obligations.
by Philip Koltun

31 **Deployment Optimization for Embedded Flight Avionics Systems**
Intelligent algorithms can be used to refine system deployments to reduce system cost and resource requirements, such as memory and processor utilization.
by Brian Dougherty, Douglas C. Schmidt, Jules White, Russell Kegley, and Jonathan Preston

CrossTalk

OUSD(AT&L) Stephen P. Welby
NAVAIR Jeff Schwalb
DHS Joe Jarzombek
309 SMXG Karl Rogers

Publisher Justin T. Hill
Advisor Kasey Thompson
Article Coordinator Lynne Wade
Managing Director Brent Baxter
Managing Editor Brandon Ellis
Associate Editor Colin Kelly
Art Director Kevin Kiernan

Phone 801-775-5555
E-mail stsc.customerservice@hill.af.mil
CrossTalk Online www.crosstalkonline.org

CrossTalk, The Journal of Defense Software Engineering is co-sponsored by the Under Secretary of Defense for Acquisition, Technology and Logistics (OUSD(AT&L)); U.S. Navy (USN); U.S. Air Force (USAF); and the U.S. Department of Homeland Defense (DHS). USD(AT&L) co-sponsor: Deputy Assistant Secretary of Defense for Systems Engineering. USN co-sponsor: Naval Air Systems Command. USAF co-sponsor: Ogden-ALC 309 SMXG. DHS co-sponsor: National Cyber Security Division in the National Protection and Program Directorate.

The USAF Software Technology Support Center (STSC) is the publisher of CrossTalk providing both editorial oversight and technical review of the journal. CrossTalk's mission is to encourage the engineering development of software to improve the reliability, sustainability, and responsiveness of our warfighting capability.

Subscriptions: Visit <www.crosstalkonline.org/subscribe> to receive an e-mail notification when each new issue is published online or to subscribe to an RSS notification feed.

Article Submissions: We welcome articles of interest to the defense software community. Articles must be approved by the CrossTalk editorial board prior to publication. Please follow the Author Guidelines, available at <www.crosstalkonline.org/submission-guidelines>. CrossTalk does not pay for submissions. Published articles remain the property of the authors and may be submitted to other publications. Security agency releases, clearances, and public affairs office approvals are the sole responsibility of the authors and their organizations.

Reprints: Permission to reprint or post articles must be requested from the author or the copyright holder and coordinated with CrossTalk.

Trademarks and Endorsements: CrossTalk is an authorized publication for members of the DoD. Contents of CrossTalk are not necessarily the official views of, or endorsed by, the U.S. government, the DoD, the co-sponsors, or the STSC. All product names referenced in this issue are trademarks of their companies.

CrossTalk Online Services:
For questions or concerns about crosstalkonline.org web content or functionality contact the CrossTalk webmaster at 801-417-3000 or webmaster@luminpublishing.com.

Back Issues Available: Please phone or e-mail us to see if back issues are available free of charge.

CrossTalk is published six times a year by the U.S. Air Force STSC in concert with Lumin Publishing <luminpublishing.com>. ISSN 2160-1577 (print); ISSN 2160-1593 (online)

CrossTalk would like to thank
309 SMXG for sponsoring this issue.

The Publisher's Choice Issue

It was an absolute pleasure to create this special, "Publisher's Choice" issue. By selecting this theme, we have the unique opportunity at CrossTalk to publish some phenomenal articles that have been waiting far too long to be published. One often-overlooked difficulty in publishing is that so many great articles are received; yet we do not have enough space to share them all. This issue has given us the chance to go back and revisit the "best of the rest" that have yet to be in print from all realms of the software engineering field.

I would also like to take this opportunity to give special thanks to all the wonderful authors who support CrossTalk. There are so many great authors who dedicate countless hours of time and effort in providing us with such high quality content we see in each issue. To all of you who have taken the time to submit your work to us, for no other compensation than our gratitude, we give you our sincere thanks. Furthermore, all of us here at CrossTalk would like to sincerely thank our readership! We exist to serve your needs. We encourage all of you to write to us with suggestions for themes, layout and design ideas, letters to the editor, and any other comments you may have so that we can continue to improve and deliver the best issues possible.

We begin this issue with an article that has been waiting far too long to be published. Craig A. Lee and Samuel D. Gasster share their insights into the use of proxies to incorporate on-orbit sensors into netcentric environments in *Netcentric Proxies for On-Orbit Sensors*. Next, Felix Bachmann gives us an interesting analysis as to how major system issues can be avoided by incorporating ATAM style design peer reviews in *Give the Stakeholders What They Want: Design Peer Reviews the ATAM Style*.

Michael Tarullo shares his knowledge in bridging the gap between theory and practice as to how sound software architectures can be produced consistently and practically in *Software Architecture Theory and Practice.* Madhav S. Phadke and Kedar M. Phadke tackle the problem of massive cost and market delays of technology due to testing by exploring the benefits of using Orthogonal Arrays for generating test plans in IT systems. The methodology and results presented in their article, *Utilizing Design of Experiments to Reduce IT System Testing Cos*t, may indicate that a better methodology may be on the horizon.

T. R. Gopalakrishnan Nair and Suma. V give us an in-depth look at how advancements in software engineering practices enables the development of more cost effective and quality products through advanced defect management strategies in *Defect Management Using Depth of Inspecting and the Inspection Performance Metric*. Philip Koltun shares his thoughts on the benefits and constraints of Free and open Source Software, as well as the many resources available for optimal utilization in *Free and Open Source Software Use: Benefits and Compliance Obligations*.

To conclude the issue, we are featuring the article *Deployment Optimizing for Embedded Flight Avionics Systems*, a collaborative work by five esteemed authors describing the benefits of intelligent algorithms to reduce cost and resource requirements in refined system developments. As always, we have included a humorous yet insightful BackTalk entitled *Geek Mystique* by Kasey Thompson.

To the authors, we work diligently to share your ideas with our readers and truly appreciate all of your time and effort in sharing this valuable information to the software community. To our readers, thank you for your continued support and hope that we continue to exceed expectations by publishing the highest quality articles.

Justin Hill
Publisher

Netcentric Proxies for On-Orbit Sensors

Craig A. Lee, The Aerospace Corporation
Samuel D. Gasster, The Aerospace Corporation

This paper investigates the use of proxies to incorporate on-orbit sensors into netcentric environments. Proxies can provide a natural system interface that observes all of the tenets of netcentricity. Proxies can provide support for security, policy enforcement, reliability, mediation, power, performance, and operational management. Proxies can also support information assurance by providing a means to enforce the separation of system components based on security policy and practices. Proxies could even be used to determine the "personality" or "look and feel" of how on-orbit resources are exposed to external clients.

1 Introduction

With the growing influence of netcentricity, there is a desire in the space community to apply it to all system elements. Netcentricity entails information and services that can be discovered through a standardized messaging protocol and used by any person or system with the right authentication and authorization. Such netcentric operations are commonly supported through some type of SOA using the appropriate vocabularies, metadata schemas, and ontologies. When properly implemented, this approach can provide much better system extensibility and interoperability, and help avoid stove-piped systems and vendor lock-in.

However, not all system elements, such as on-orbit sensors, are amenable to direct exposure in a netcentric system. Since there is a fundamental trade-off between performance and flexibility, any system that must operate in a specialized, resource-constrained environment may not be able to fully support the requirements for netcentric communication and interaction. Furthermore, sets of on-orbit sensors may be part of a larger system that must be managed as a whole. For instance, any single on-orbit sensor may be on a vehicle with other sensors that interact and share local resources. Any single sensor could also reside on a module that is part of a fractionated cluster in which various modules are sharing resources [1]. Each vehicle could be in a constellation of vehicles that must be managed as a whole at some level.

Hence, in this paper, we will investigate the use of netcentric proxies to make on-orbit sensors available in a general service-oriented architecture, while transparently managing the constrained bandwidth, latency, orbital connectivity, and functional characteristics in an intelligent manner. That is to say, the netcentric proxy can actually expose the control and data of individual sensors to external users, but it can also expose an abstraction or higher-level interface to the sensor that is more appropriate and simpler for external users. We also note that netcentric proxies can also be used to virtualize on-orbit sensors, since users would not have to communicate with a specific hardware device at a fixed address, but could communicate through any instance of the appropriate proxy.

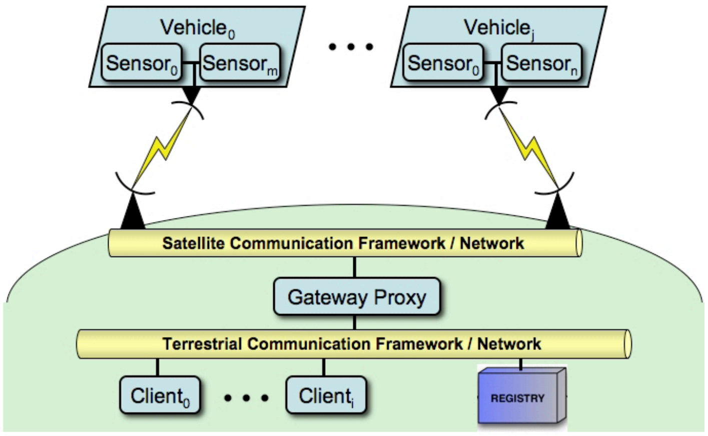


Figure 1. A Notional Satellite Gateway Proxy Architecture

2 Proxies for On-Orbit Sensors

As noted already, not all system elements are suitable to a netcentric SOA environment. On-orbit sensors operate in a highly constrained environment. On-board power is limited, communication is through highly specialized radio frequency (RF) and optical links, connectivity can be intermittent, and unique operation and usage policies are strictly enforced. On-orbit sensors, and their data, may also be classified at a higher level than other system components. The appropriate protections should be in place as data moves from the space segment, through the space-to-ground link, and into the ground segment.

For all of these reasons, directly incorporating on-orbit sensors in a netcentric SOA would be very problematic. SOAs typically require a common transport layer for communication (such as TCP/IP), services that "come and go" could cause disruption for clients, and an on-board sensor is probably not the place to enforce policy across competing requests from a potentially large number of clients. To deal with any of these issues on-board would require more on-board computing and power demand just to do "housekeeping".

It would be possible, however, to indirectly incorporate on-orbit sensors into a netcentric SOA by making them "available" through one or more gateway proxies. Figure 1 illustrates how such gateway proxies could be used to do this. Here a gateway proxy provides one or more services using a terrestrial SOA. This proxy is also connected to a satellite communication system that has the physical uplinks to vehicles and their sensors. These uplinks provide connectivity to multiple vehicles, each of which may have multiple sensors.

With this notional architecture, many issues and capabilities can be addressed, which we discuss in the following subsections.

2.1 Protocol Conversions

Proxies are a natural place to do protocol conversions between terrestrial networks and on-orbit vehicles since they are, by definition, in between the two. (This is called mediation in the parlance of netcentricity.) While work has been done in running common network protocols, such as TCP/IP, over delay tolerant networks, i.e., on an interplanetary scale, this will not be common. Existing systems will use unique, specialized communication and interaction protocols that are very different from those used in SOAs.

2.2 Addressability of Individual Sensors

In networks and SOAs, being able to address and send a message to specific recipients is a fundamental capability. Names and addresses of vehicles and sensors could be published to the registry that external systems are allowed to discover and use. Behind the gateway, however, these names and addresses could be mapped to whatever scheme makes the most sense internally.

2.3 Higher Level System Services

We also note that the externally visible names and addresses could, in fact, represent not just individual sensors, but also aggregate functionality provided by the sensors, vehicles, clusters, or an entire constellation. The use of gateway proxies would allow a range of services to be exposed on the terrestrial SOA—from individual sensors to higher level, aggregate services that define the apparent "behavior" or "personality" of the entire satellite system. As an example, a user may want infrared (IR) surveillance data with specific performance parameters. They could submit a request to an IR surveillance service that determines how best to satisfy this request with the available resources. The user could get an initial report describing how the request will be met, and if adequate, the user could resubmit the request for the actual data.

2.4 Managing Orbital Connectivity

Depending on the presence of cross-links in a particular satellite constellation, vehicles and their sensors may only have periodic connectivity to the ground. A gateway proxy could provide a continuous presence for the sensors, even if they are not over a ground station, thus providing a more robust client interface and experience.

2.5 Managing Reliability

Beyond just orbital connectivity, proxies could manage all aspects of the externally perceived reliability for sensors and vehicles. If the gateway can communicate with more than one ground station, it can reroute traffic if one ground station fails, or if the network link fails. In the event of a failure somewhere in the system the proxies could, at a minimum, provide information to clients about the failure. It can also attempt to transparently shield the client from failure by looking at alternate ways to satisfy service requests.

2.6 Power Demand

Some client service requests will require the expenditure of power onboard one or more satellites. The aggregate of client requests may, in fact, exceed the available on-board power. Hence, the gateway proxy would be where the best location to enforce energy policies could be enforced. By examining the client request stream, the proxy could rearrange or delay requests, when possible, to avoid excessive power demands.

2.7 Security and Information Assurance

Gateway proxies are also the natural "gatekeepers" for the on-orbit assets. They can fully participate in the SOA's security mechanisms and support information assurance. Clients must authenticate to the gateway to establish their identity and be authorized to request services and data from the satellites. Access is based on the client's role within mission operations and pre-defined usage policies. Proxies can also provide encryption, checksums, and other methods for monitoring data integrity.

2.8 Operational Policy Enforcement

Beyond issues of power management, security and information assurance, gateway proxies are also the place where all operational policies concerning on-orbit assets could be enforced. For example, proxies could enforce an operational policy of "do not slew the bore-sight of the sensor across the disk of the sun," or "do not exceed a given power/duty cycle," etc. Clearly proxies could be the policy enforcement point as part of an overall resource management and scheduling system.

Homeland Security

The Department of Homeland Security, Office of Cybersecurity and Communications, is seeking dynamic individuals to fill several positions in the areas of software assurance, information technology, network engineering, telecommunications, electrical engineering, program management and analysis, budget and finance, research and development, and public affairs. These positions are located in the Washington, DC metropolitan area.

To learn more about the DHS Office of Cybersecurity and Communications and to find out how to apply for a vacant position, please go to USAJOBS at www.usajobs.gov or visit us at www.DHS.GOV; follow the link Find Career Opportunities, and then select Cybersecurity under Featured Mission Areas.

2.9 Cluster and Constellation Configuration

Multiple sensors, modules, and vehicles may not be independent of each other and may have to be managed as a unified system. All sensors on a particular vehicle are related, since they must share common resources, e.g., power, communication bandwidth, etc. They may also be related through configurable functional attributes that are sensor-specific, e.g., band, filtering, etc. Hence, while clients may want to interact with individual sensors or with higher-level aggregate services, the proxy may have to manage the sensors as a group. (Essentially, this is enforcing configuration policy.)

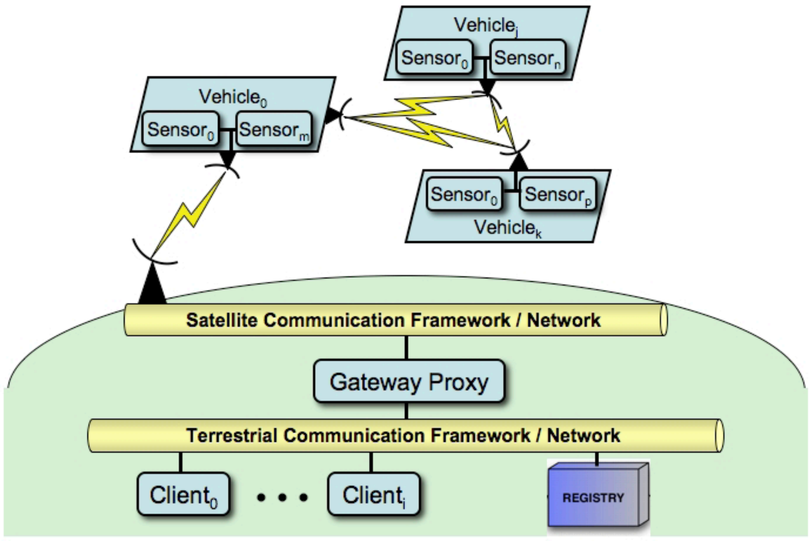


Figure 2. Gateway Proxy to Satellite Cluster Configuration

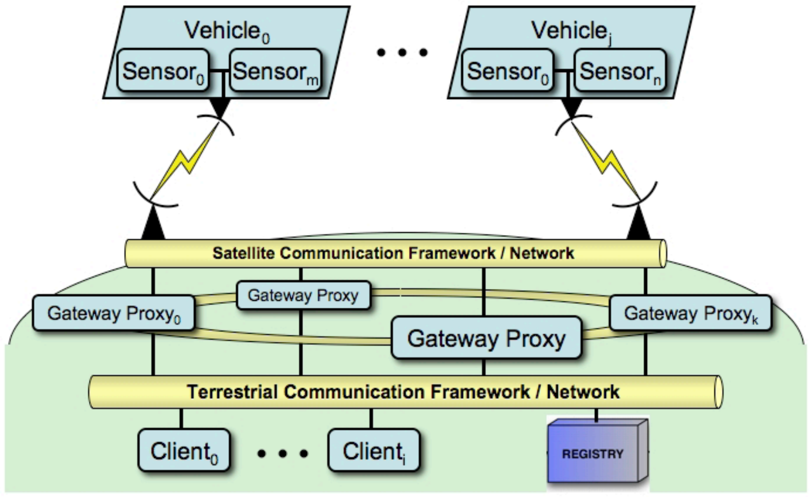


Figure 3. Satellite Gateway Proxies in a Peer-to-Peer Configuration

Likewise, sets of vehicles may have to be managed as a whole. Vehicles could be in a leader-follower configuration, a group or cluster of satellites, or in a multiple plane constellation. A specific example of satellite clusters is the DARPA System F6 program [2]. The idea behind F6 is to develop satellite architectures consisting of "future, flexible, fast, fractionated, free-flying spacecraft united by information exchange." This is illustrated in Figure 2, where a set of vehicles is in cluster flight configuration and communicating through their own RF cross-links. Each vehicle is a fractionated module with a specific set of functions provided for the cluster, i.e., the entire satellite system. When under attack, such modules can disperse and reform the cluster at a later time when it is safe to do so. If any one module fails, its functions could be taken over by another until a replacement module is available. Proxies would be very useful for interfacing such fractionated satellite architectures with a terrestrial service architecture.

2.10 Peer-To-Peer (P2P) Network of Gateway Proxies

In the discussion so far, we have presented the gateway proxy as if it were a single point of entry. The gateway could, in fact, have more than one "point of entry." There could be a P2P network of gateway proxies, as illustrated in Figure 3. Clients could contact the closest peer when requesting data or services from the on-orbit assets. (See "Terrestrial Data Archives.") The gateway peers could also provide redundancy and continuity of operations, i.e., reliability. The peers could be physically separated from one another such that if one peer crashes or is off-line for any reason, then access to the on-orbit assets is still possible by re-routing through another peer. One could also deploy a peer downrange on the battlefield to serve as the battlespace local point of contact.

2.11 Terrestrial Data Archives

Satellite sensors can produce tremendous amounts of data that must be served to clients and archived for future use. Such archives may be behind the gateway proxies or anywhere on the terrestrial SOA. If the archive is on the terrestrial SOA, the gateway could at least act as the agent that provides data to the archive.

If the archive is behind the gateway, however, then the gateway can serve as the gateway to the data archive as well. That is to say, the gateway could enforce data policy by managing access to the data, replicating data to different sites for faster access and reliability, and even providing data virtualization services. When a client requests satellite data from the gateway, it first looks to see if the requested data is available in the archive. If the requested data products are not available, then the gateway could actually schedule the on-orbit sensor to collect the raw data necessary to satisfy the client request.

2.12 Managing a Larger Sensor Network

Finally, we note that the on-orbit sensors could actually be part of a larger sensor network supplying data and information to a wide set of consumers. Consumers may want to interact with all of their data providers through a uniform model and interface to improve ease of use. On-orbit sensors may be only one of many data providers. Such an interface could define uniform ways for requesting data, specifying when the sensor produces data, and how the data is reported. One possible standard relevant to such sensor networks is the Sensor Web Enablement standard from the Open Geospatial Consortium [3].

3 Summary, Discussion, and Future Work

We have presented the concept of using proxies to manage the exposure of on-orbit vehicles and sensors in netcentric systems. Building on established concepts in computer networks and distributed systems, we argue that proxies on SOA—as an intermediary between clients and on-orbit assets—provide a mechanism to implement a wide range of important and useful capabilities. These capabilities include information assurance, policy enforcement, reliability, mediation, power, performance, and operational management. This can also be extended to managing how the "personality" or "look and feel" of vehicles and sensors are presented to external clients.

The concept of netcentric proxies for on-orbit vehicles and sensors has significant value, but clearly more thorough studies should be done to evaluate the possible difficulties of implementation and the actual benefits. For any specific systems, the general issue of increased latency introduced by a proxy would have to be evaluated. Also, any implementation in a real-world satellite system would carry with it any number of conflicting goals and design compromises. These conflicting goals and design compromises may have nothing to do with SOAs or proxies, but may impact their overall effectiveness.

To avoid pitfalls, it is clear that prototyping programs should be undertaken that start small and incrementally build capabilities for evaluation. The capabilities identified could be partitioned into phases that build on one another. Such prototypes could possibly leverage the Netcentric Core Enterprise Services [4] that are already being developed by the Defense Information Standards Agency (DISA). In addition to the engagement with DISA, the notion of netcentric proxies could also be promoted in defense contractor and community organizations, such as the Network-Centric Operations Industry Consortium [5], the Ground System Architectures Workshop [6], and the Federal SOA Community of Practice [7]. This would facilitate "closing the loop" among user/government requirements, standards organizations, and the vendor community.

Acknowledgments:

This work was supported by The Aerospace Corporation through the Innovation Grant Program of the Research and Program Development Office.

ABOUT THE AUTHORS

Dr. Craig A. Lee is a Senior Scientist at The Aerospace Corporation and current serving as President of the Open Grid Forum. Dr. Lee has worked in the area of parallel and distributed computing for the last 30 years. He has conducted DARPA and NSF sponsored research and served as a review panelist for the NSF, NASA, DOE, and INRIA. He has published over 60 technical works and sits on the editorial board of two journals. Dr. Lee holds a Ph.D. in Computer Science from the University of California, Irvine.

Computer Systems Research Department
The Aerospace Corporation, P.O. Box 92957
El Segundo, CA 90009
E-mail: lee@aero.org

Dr. Samuel Gasster is a Senior Scientist at The Aerospace Corporation, where he specializes in the application of high performance computing technology for scientific and remote-sensing applications, data-modeling and data-management system development, and systems and software engineering. He has worked at Aerospace for over 20 years and has supported a wide range of defense and civilian programs and agencies, including the USAF, NASA, NOAA, and DARPA. He currently supports the DARPA System F6 Program. His research interests include Quantum Information Science and Technology, new approaches to space mission systems engineering and complex systems. He has taught remote sensing and computer science courses at UCLA Extension and the Aerospace Institute. He holds a Ph.D. in physics from the University of California, Berkeley and an S.B. in mathematics from MIT.

Computer Systems Research Department
The Aerospace Corporation, P.O. Box 92957
El Segundo, CA 90009
E-mail: gasster@aero.org

REFERENCES

1. O. Brown and P. Eremenko. The value proposition for fractionated space architectures. AIAA Space 2006, (AIAA-2006-7506), 2006.
2. DARPA. System F6. <http://www.darpa.mil/TTO/ Programs/sf6.htm>.
3. The Open Geospatial Consortium. SensorWeb Enablement. < http://www.opengeospatial.org/projects/groups/sensorweb>.
4. Defense Information Systems Agency. Net-Centric Enterprise Services. < http://www.disa.mil/nces>.
5. The Network-Centric Operations Industry Consortium. < http://www.ncoic.org>.
6. Ground System Architectures Workshop. < http://sunset.usc.edu/GSAW>.
7. Federal SOA Community of Practice. < http://semanticommunity.wik.is/Federal_SOA_Community_of_Practice>.

Give the Stakeholders What They Want: Design Peer Reviews the ATAM Style

Felix Bachmann, Software Engineering Institute

Abstract. The Architecture Tradeoff Analysis Method® (ATAM®) is used to evaluate the architecture of a software intensive system to determine if it meets the organization's business and mission goals [1]. ATAM is typically applied at the end of the architecture design process. Looking at the results of many past ATAM evaluations, it becomes apparent that some mechanism is needed to better guide the architecture design process. Many ATAM evaluations show major issues with the system that could have been avoided with the right design approach.

In a recent project, SEI guided architects of an organization through the design of a major system for the financial market. We used a Quality Attribute Workshop (QAW) to generate the first set of important quality attribute scenarios that interested stakeholders. The design was created utilizing the SEI's Attribute Driven Design method (ADD) [2] in which the generated scenarios were transformed into an architecture design. On a bi-weekly basis, ATAM style peer reviews were conducted with the architects to ensure that the design actually addressed the requirements. This combination of methods achieved some interesting results for this project:

* First, an architecture evaluation using ATAM conducted at the end of the architecture design process was completed in half the time than comparable ATAMs done on large software systems and did not uncover any unexpected risks. In short, the evaluation showed that the system indeed provides what the stakeholders want.

* Secondly, the ADD method combined with the ATAM style peer review made the architecture design tasks transparent for both project management and the stakeholders. Instead of trying to explain architecture diagrams, scenarios (from the QAW) with their associated risks (from the ATAM style peer review) were reviewed on a biweekly basis. Seeing the risks being mitigated over time convinced the stakeholders that the project was on the right track.

* Finally, the architecture team never had to be pushed to document their architecture. Just the fact that the architects had to prepare for the biweekly peer reviews was sufficient incentive to write down how the current design would fulfill the stakeholder scenarios. Basically, the architecture documentation was created continuously during the design with no additional effort.

Quality Attribute Scenarios

Properly designing software architecture means that, aside from the necessary functionality, the system will meet the required quality attribute requirements such as modifiability, interoperability, and security, to name just a few [3]. In fact, it is the system's architecture that determines if the system meets the quality attribute requirements. That is why SEI created architecture methods like QAW [4] and ADD [5], as well as the ATAM [2], that are centered on utilizing quality attribute scenarios as a more precise way of specifying the quality attribute requirements a system has to fulfill.

During a QAW, the stakeholders articulate and prioritize quality attribute scenarios based on their business and mission goals. It is the architect's job to then take those scenarios and transform them into a design that will support these goals. This is exactly what SEI's ADD method is used for. Performing an ATAM at the end of the design process involves reviewing the quality attribute scenarios again, verifying with the stakeholders that the scenarios are still valid, and then verifying that the architecture supports these scenarios.

This sounds like a valid process, but a surprising number of ATAMs reveal that the architecture does not fulfill the requirements. Independent of the reasons why, in many cases there is no time in the schedule to actually go back and redesign the architecture. The only remaining alternatives are to either end the project or to move forward with a system designed with inherent risk hoping that nothing bad will happen.

ATAM Style Design Peer Reviews

During the design process there have to be some checkpoints that allow verification that the design will fulfill the stakeholder's expectation. Conducting peer reviews is a common method for doing so. As the ATAM results show, in many cases those peer reviews do not ensure that the appropriate system is developed. To overcome this weakness we introduced a peer review process that utilizes the same techniques ATAM evaluation uses. This makes the design process similar to Kent Beck's Test Driven Development (TDD) [6]. In a test-driven development, tests are created first, then the part of the system that is executed by the test is developed and then the tests are run. If a test fails, the developed code is corrected and the test is run again. These steps are repeated until all tests pass.

In an architecture design, the tests are actually the quality attribute scenarios. The architecture design must fulfill those scenarios to be accepted by the stakeholders as a good design. As was stated above, the scenarios are already defined during a QAW before the architecture is designed. The architects ensure that the current design is checked in a periodic fashion to see if the scenarios are continuing to be fulfilled. Running tests on the current design means performing a peer review using the techniques of the ATAM.

The timing and the scope of the peer review strongly depend on system complexity and the quality attribute scenarios. In our case, we decided to conduct an ATAM style design peer review every two weeks. We allocated three hours for the review and we were able to review two scenarios during those three hours. At the beginning of a two-week cycle, the architects decided on the two scenarios to focus on for that cycle. The architects then had two weeks to design the system that would support those two scenarios. During that time period, the architects had to produce a documented design including evidence that the design was appropriate and that past scenarios, already checked in earlier reviews, were not now being violated by the updated design. At the end of the cycle, a peer review was conducted with ATAM-trained SEI architects.

Steps of the ATAM Style Peer Review

An ATAM-style peer review is done by building a review team that consists of the system's architecture team and two other architects to act as reviewers. In our case we used architects from SEI, but any architect, not involved in the project and knowledgeable in the ATAM method would be able to do the review.

It is the responsibility of the architecture team to provide all necessary documentation to reviewers at the beginning of the review, explaining why the chosen scenarios are well supported. Typically the architecture team leader is the main speaker, but the other architects also provide information whenever necessary. At least one architect needs to have a good understanding of what the stakeholders actually meant when they created the scenarios. This knowledge helps to identify when a scenario was written ambiguously.

The reviewers' main responsibility is to ask questions that help the architecture team uncover issues in their design. One of the reviewers acts as a facilitator, responsible for guiding the whole review team through the review process. The other reviewer acts as a scribe, writing down the approaches, risks and the to-do items.

Let us have a more detailed look into the ATAM style design peer review process.

Step 1: Select the scenario to analyze.

A design peer review needs to have a clear focus. Instead of analyzing the whole architecture—which could be a very tedious task—the review only needs to uncover the risks associated with one scenario. The peer review starts by selecting the scenario to review. This is usually one of the scenarios selected at the beginning of the two-week design cycle, but could also be any other scenario. It often happens that, during the design process, a scenario will get refined into multiple, more detailed scenarios addressing different aspects of the requirements. For example, the starting scenario could have been one stating that the system has to be available 24 hours, seven days per week. During the design process, the architects may have discovered that hardware failures and software failures need to be treated differently. Therefore, the availability scenario might have been broken into two scenarios, each addressing different aspects of availability.

As a rule of thumb, we saw that every scenario created by stakeholders during a QAW was typically divided into three to five more specific scenarios.

A design peer review needs to have a clear focus. Instead of analyzing the whole architecture—which could be a very tedious task—the review only needs to uncover the risks associated with one scenario.

Step 2: Elicit the architecture approaches.

An architecture approach is a pattern or tactic [2] used in the architecture to support the chosen quality attribute scenario. On one hand, eliciting the approaches allows reviewers to very quickly see if there is sufficient support for that scenario. On the other hand, it also allows the reviewers to ask questions about the possibly negative consequences the approach has on other scenarios. The scribe writes down the approaches including the rationale on why they were chosen.

Step 3: Analyze architecture approaches.

The analysis of the architecture approaches is done as a question and answer session where the reviewers ask questions about the solution and the architects answer the questions by pointing to the parts of the architecture documentation that provide the answers. Here are clues about how to treat the answers:

* If a question cannot be answered, the scribe writes it down as a risk.
* If the provided answer is problematic because it might violate some other scenarios, it is written down as a risk.
* If the answer is that this is still an open issue, it is written down as a to-do item.
* If the answer satisfies the reviewers, it is written down as evidence with a pointer to the supporting documentation if it was not already done. The scribe also notes every piece of documentation, such as structural diagrams (module views, component and connector views, deployments, views, etc.) or behavioral diagrams (sequence charts, state diagrams, etc.) that was used during the review of the scenario.

Step 4: Review results.

Step 3 usually results in a list of five to 10 risks per scenario. This may sound like a big number, but this level is normal when the scenario is reviewed for the first time. It also means that some sort of redesign has to follow. In Step 4 the architecture team and the reviewers analyze the captured list of risks as well as to-do items and decide on appropriate actions. The best case would be that the scenario is solved and no further action is required. More commonly, the appropriate actions are to adjust the architecture, to build a prototype to provide better insights, or have a discussion with the stakeholders because the scenario might be impossible to achieve.

After Step 4, the architecture team should have a clear view about what to do next with the reviewed scenarios.

Performing ATAM style design peer reviews every second week was never seen as a burden by the architects. They were actually looking forward to the next review because the reviews provided them with valuable input and they could see progress when the list of risks and the to-do list became smaller and smaller over time.

Conclusion

In our experience, performing ATAM style design peer reviews every second week was never seen as a burden by the architects. They were actually looking forward to the next review because the reviews provided them with valuable input and they could see progress when the list of risks and the to-do list became smaller and smaller over time. The architects also saw the value of early feedback. Even if they went down a wrong path, the most time they would lose was two weeks.

These benefits are apparent, but there were other positive side effects. The peer reviews trained the architects to think in terms of uncovering risks and mitigating them. This enabled the architects to have more productive discussions with stakeholders, such as the project manager or the program office about their requirements using scenarios with their attached risks. When product development goes into the architecture design phase, outsiders often perceive that nothing is happening even if the architects show diagrams and pictures. For an outsider those pictures do not mean anything, but talking about scenarios and risks makes the whole architecture design process transparent. Even if you do not understand what those architecture diagrams mean, you can clearly track risks and see progress as those risks are slowly being mitigated.

We also did an ATAM at the end of the architecture design, just to make sure that nothing was missed. The ATAM was done by a completely independent SEI team that was not involved in the design. Since the architecture team was able to provide to the ATAM team all the artifacts that are usually created during Phase 1 of the ATAM, the ATAM team could focus on Phase 2 only. This expedited verification with the stakeholders that indeed the architecture fulfilled their needs. The result was that the ATAM did not find any unknown or unaddressed risk. Basically, the ATAM acknowledged that the stakeholders would get the system they want.

It is also noteworthy that in the project plans there was never a "documentation" task. The fact that the architects had to deliver proof every two weeks automatically led them to document their design immediately. They knew that if an important concept was not written down it would end up as a risk.

Disclaimer:

Architecture Tradeoff Analysis Method® and ATAM® are registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

ABOUT THE AUTHORS

Felix H. Bachmann is a Senior Member of the Technical Staff at the Software Engineering Institute (SEI) working in the Architecture Centric Engineering Initiative. He is co-author of the Attribute-Driven Design Method, a contributor to and instructor for the ATAM Evaluator Training, and a co-author of Documenting Software Architectures: Views and Beyond.

Before joining SEI he was a software engineer at the Robert Bosch GmbH in Corporate Research, where he worked with software development departments to address the issues of software engineering in small and large embedded systems.

Software Engineering Institute
Felix Bachmann
4500 5th Avenue
Pittsburgh, PA 15213
Phone: (412) 268-6194
Fax: (412) 268-6257
E-mail: fb@sei.cmu.edu

REFERENCES

1. Clements, P.; Kazman, R.; Klein, M. Evaluating Software Architectures: Methods and Case Studies Addison Wesley, 2002. Engineering Institute, Carnegie Mellon University, 2002.
2. Bass, L., Clements, P., and Kazman, R. 2003. Software Architecture in Practice, Second Edition. Boston, MA: Addison-Wesley.
3. Bachmann, F.; Bass, L.; Klein, M. Illuminating the Fundamental Contributors to Software Architecture Quality (CMU/SEI-2002-TR-025). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2002.
4. Barbacci, M.; Ellison, R.; Lattanze, A.; Stafford, J.; Weinstock, C.; Wood, W. Quality Attribute Workshops (QAWs), Third Edition (CMU/SEI-2003-TR-016). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2003.
5. Wojcik, R.; Bachmann, F.; Bass, L.; Clements, P.; Merson, P.; Nord, R.; Wood, W. Attribute Driven Design (ADD), Version 2.0 (CMU/SEI-2006-TR-023). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2006.
6. Beck, K. Test-Driven Development: By Example. Addison-Wesley, November 2002.

Software Architecture Theory and Practice

Michael Tarullo, L-3 Communications

Abstract. There is often a gap between widely accepted software engineering theory and practice. This is also true for the concept of software architecture. While the concept of software architecture has been in existence for quite some time, there is still a great deal of confusion over just what software architecture actually is. Moreover, lack of a clear understanding of the concept of software architecture makes it extremely difficult to work with pragmatically. This article attempts to show how sound software architectures can be produced quite practically and documented consistently. A definition of software architecture is adopted and a model for creating software architectures by using the de-facto standard software engineering modeling tool, UML (v2.0), is introduced.

Introduction

In many fields, there is often a gap between theory and practice. Software engineering is no different. Misconceptions about software architecture, particularly by practitioners, make it difficult to communicate software architectures effectively. SEI's website [1] demonstrates the astounding diversity that exists with respect to the definition of software architecture. This website lists two modern definitions, eight classical definitions, 18 bibliographical definitions and numerous community definitions. The first three categories indicate a general agreement on the definition of the term by theoreticians and academicians. It is the wide variety of definitions held by those in the last category that is troubling, specifically because they appear to represent practitioners. And such confusion can make it difficult, if not impossible, to use the concept in a practical fashion.

This article attempts to show how sound software architectures can be produced quite practically, in a repeatable and understandable fashion, by adopting a widely held definition for the concept of software architecture, adopting a model for creating software architectures, and by using the de-facto standard software engineering modeling tool, UML (v2.0), to convey a software architecture.

Theory

Kruchten, et. al. [2] provide an excellent presentation on the history of software architecture. Their paper traces the development of software architecture theory from the time the paper was published back to its origins and before. Mary Shaw and David Garlan [3] published one of the earliest books on the subject. It is fitting that they begin their book with the question, "What is Software Architecture?"

Both the theory and practice of software architecture must be rooted in a clearly expressed and universally accepted definition of the term. What is needed is a definition that succinctly and cogently expresses the concept of software architecture. Moreover, such a definition must express the concept in such a way that it can be used practically. We turn to the myriad of definitions compiled by SEI to extract the essence of the meaning of software architecture. Many if not most of the definitions published on the SEI website have three things in common; 1) organization of a system, 2) components, and 3) relationships. While there are many other concepts conveyed, it is these three terms, or synonyms thereof, that persist throughout the definitions provided and are at the core of the theory. As a result, the definition provided by Bass, et. al. [4], that is, "The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them," is adopted herein since it contains each of the three common concepts cited above and complements the techniques that can be used for creating the software architectures described below. It is assumed that "structure" and "elements" in Bass's definition are synonyms for organization and components respectively, as used on SEI's website.

Also needed to build sound, practical software architectures, is a theoretical model that places software architecture within the larger context of software design. Such a model is provided by Mowbray and Malveau [5]. Their Scalability Model (SM) represents the software design continuum as a series of design levels, each representing the software under consideration at a different level of abstraction. Mowbray and Malveau describe each of these levels thus:

- The global level is concerned with the design issues that are applicable across all systems (enterprises).
- The enterprise level is focused upon coordination and communication (of systems) within a single organization.
- The system level deals with the coordination and communication across applications (and libraries) and sets of applications (and libraries).
- The application level is focused upon the organization of applications developed to meet a set of user requirements.
- The macro component level is focused on the organization and development of application frameworks.
- The micro component level is centered on the software components that solve recurring software problems.
- The classes level is concerned with the development of reusable objects and classes.

While their model was created to provide the foundation of their work in Common Object Request Broker Architecture design patterns, it is most relevant to object-oriented software development but is certainly abstract enough to be applied to

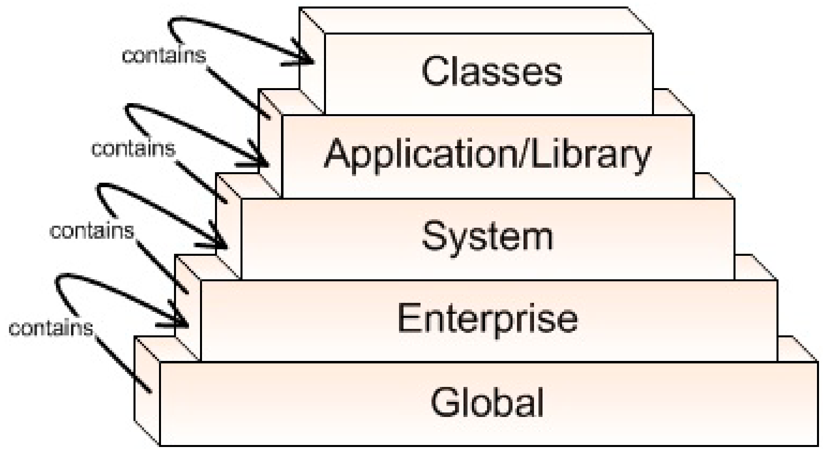


Figure 1:

other methods of software development as well. In this discussion we will use a modified version of the SM (see Figure 1) and architectural issues will focus on the global, enterprise and system levels.

Having established a sound definition of software architecture and a model for constructing such architectures, the practical application of these concepts can now be discussed.

Practice

This section will illustrate a process that can be used to build and graphically document software architectures. A discussion of the capturing of major architectural decisions is beyond the scope of this paper. For a discussion of this topic consult the work of Tyree and Akerman [6]. Also, the textual description of a software architecture that would be included in a formal software architecture document is only briefly mentioned.

This section does discuss the practical application of the modified SM shown in Figure 1. A practical application of the original model is provided by Tepfenhart [7]. This paper describes a rather strict application of UML (v2.0) to several of the concepts presented by Tepfenhart [7]. The basis for the usage of UML described here can be found in several sources [8, 9, 10].

Mowbray and Malveau [5] state that, "One of the key benefits of architecture is the separation of concerns … the SM separates concerns based upon scale of software solutions. The model clarifies the key levels inherent in software systems and the problems and solutions available at each level."

This approach fosters decomposition, a major practice used to control complexity in large (or any size for that matter) software systems. The following summarizes how the SM provides a guideline for the architectural decomposition process. This is followed by an abstract example of how UML would be used to document the global, enterprise and system levels.

In the modified SM shown in Figure 1, we treat each level as a container that holds components that are elements of the next level above. That is, the global level is a container for enterprise components, the enterprise level is a container for system components, and so on for each level of the model. It is important to note here that the term component is used throughout in both a traditional [11] (e.g. software component) and non-traditional sense. We will examine this again when describing the process used to document architectures later in this paper.

To decompose a large software system for the purpose of creating an architectural model, we start with the global level. The global level architecture is composed of enterprise components. Enterprises are the identifiable business units or organizations whose software will interact to achieve some computational goal. The business unit or organization sponsoring the software development is identified as well as business partners, customers, or suppliers—in short any business entity that may interact with the sponsoring organization.

The enterprise level consists of the organization sponsoring the software development and is composed of all the systems that will be employed to achieve the project goals and requirements. Systems identified at this level are not confined to just systems that will be developed as part of the software development effort. In-house legacy systems and COTS products, either existing or that need to be purchased as part of the present effort, are also identified. In this way no effects from unanticipated interfaces should occur during detailed design. For each system that will be developed, the architecture clearly demonstrates collaboration with other systems, either under development, already existing, or with plans to be purchased.

The next step in the process is the decomposition of the systems to be developed that were identified at the enterprise level. The components at this architectural level are either subsystems, applications or libraries. In UML v2.0 libraries are represented as artifacts. Here, we choose to represent libraries as components. Almost invariably, libraries are the manifestation of components and stereotyping components for the various levels of the SM is a natural and more than acceptable method of presentation. Applications are defined as standalone executable software components while libraries though standalone, rely on applications for their run-time execution. Libraries may be either internal or external to applications.

To graphically document a software architecture defined in this way, we use the UML component diagram. The component diagram is perfect for representing the architectural elements at each level of the SM. Furthermore, it is also a perfect companion to the component-oriented definition adopted here. Since the components at several levels of the SM are not software components as defined by the UML, we use stereotypes to indicate the components at each level. Only the components at the system level and above are software components in the sense of the UML definition. We use interfaces, direct connections, and delegation connectors to indicate the relationships between components at all levels. Direct connections are used to represent interfaces between components that may not be call level interfaces, such as shared files or non-digital medium. This is necessary because, as we have seen, not all components in the model represent software.

Figure 2 illustrates a UML component diagram at the global level for a software system under consideration. It conveys to the viewer that the global architecture consists of four enterprises. Assume that Enterprise X is the enterprise for which the software under consideration is being designed. Furthermore, assume that Enterprise W, Y and Z are not part of the same organizational unit (corporation, government agency, etc.) as Enterprise X. This component diagram clearly indicates that Enterprise X needs some functionality or data provided by Enterprise Z, and is provided by Enterprise Z through Interface Z. It also clearly indicates that Enterprise X will provide some functionality through Interface X that will be used by Enterprise Y. This component diagram also shows that Enterprise X has a relationship to Enterprise W through the direct connection XW. This component diagram shows no relationships among Enterprise W, Y and Z. This does not mean that such relationships do not exist; it only means that such relationships, if indeed they do exist, are not important to the architectural description of Enterprise X, and therefore do not need to be included.

We would now move on to a decomposition of Enterprise X into its constituent system components. The result of such a process would be a component diagram for Enterprise X like the one shown in Figure 3. This diagram conveys to the viewer that Enterprise X consists of five systems, Systems A, B, C, D and E. System A provides Interface A that is used by System C. Also, System B provides Interface B1 and Interface B2 which are used by System A and System D respectively. System A has a relationship to System E through direct connection AE. The viewer can also determine that System C is the system within Enterprise X to which access to Interface X by Enterprise Y is delegated; and that Enterprise X delegates to System D use of the external Interface Z that is provided by Enterprise Z. Furthermore, Enterprise X delegates access to Enterprise W to System E via the delegation connection to port Connection XW. It can also be seen that System A and System C are both providers and users of various interfaces, while System B is only a provider of interfaces and System D is only a user of interfaces. But more than this, the viewer knows exactly which interfaces and connections are provided by which systems and likewise which interfaces and connections are used.

Next, each system component identified at the enterprise level would be decomposed into applications and/or libraries. We will assume that analysis has shown that Systems A, B and C will be part of a new development project. Furthermore, we will assume that System D will be a COTS product and System E is a legacy system that is being retained, unchanged. For this discussion we will only describe the decomposition of System A. We also know, from the component diagram for Enterprise X, the relationships System A has to all the other systems at the enterprise level and we will discuss these as well.

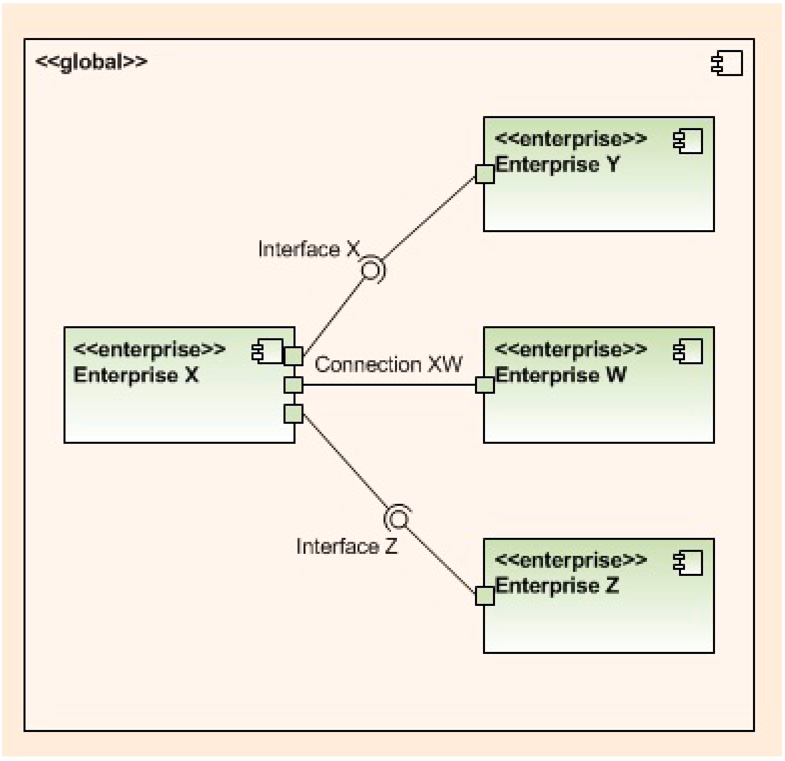


Figure 2

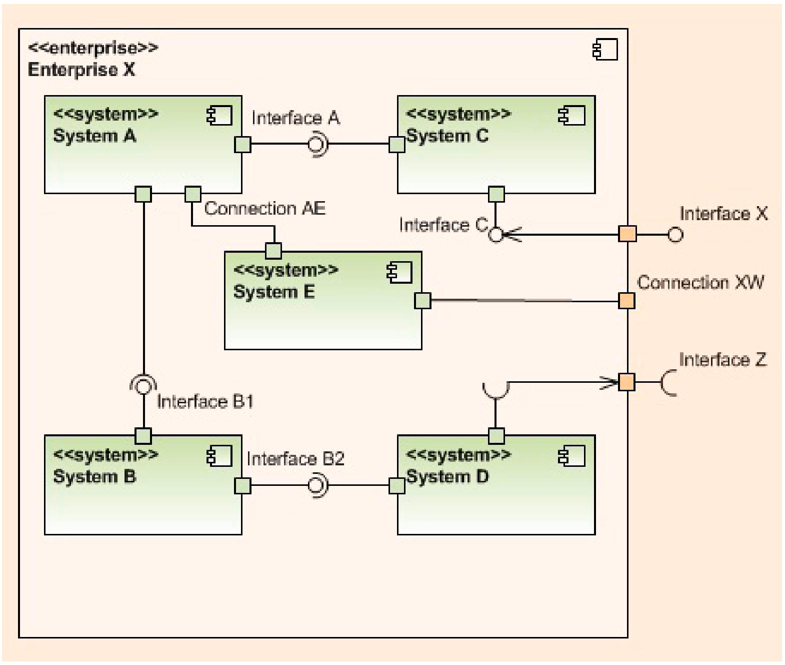


Figure 3

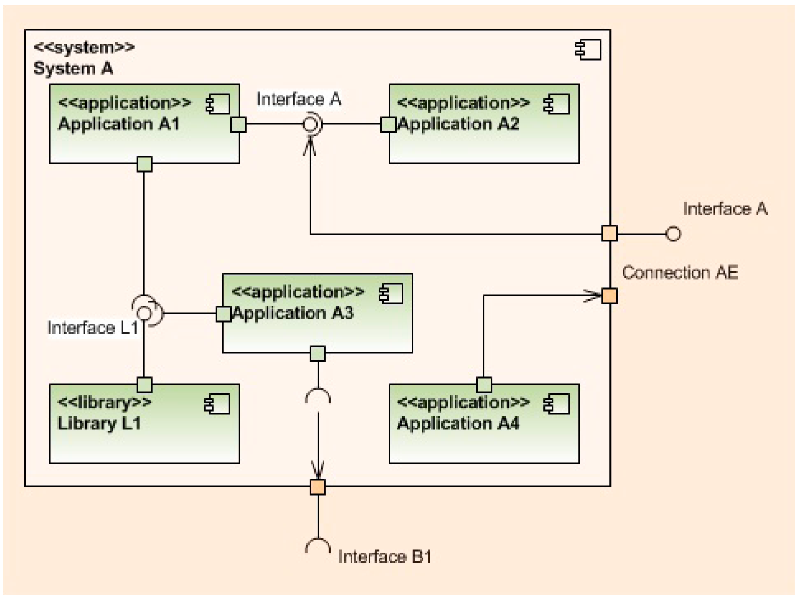


Figure 4

Figure 4 shows the decomposition of System A into its component applications and libraries. This component diagram shows that system A consists of four applications, Application A1, A2, A3 and A4 and one library, Library L1. Application A1 has a relationship to two other components that are a part of System A, that is Application A2 and Library L1. Application A1 provides Interface A that is used by Application A2. Recall that one of the system level interfaces provided by System A, and used by System C, is also Interface A. We say that System A delegates the implementation of Interface A to Application A1, as indicated in the diagram by the delegation connection that connects the external port for Interface A with the provided interface of Application A1. It should be noted that Interface A connected to the external port of System A and Interface A connected to Application A1 are in fact the same interface. Conceptually, one may think of the System A external port connection for Interface A as the access point to the interface provided by Application A1. In fact, neither System A nor Application A1 is actually capable of implementation of Interface A. Application A1 would actually delegate implementation of this interface to a specific class.

Application A1 also uses Interface L1, the provided interface of Library L1 through its required interface. Library L1 is an example of an external library component. Its implementation would be highly dependent on the programming language used to write the code for System A (e.g. in C++ on Windows it could be a Dynamic Link Library, .dll file). The concept of a library as a component more closely approximates the traditional use of the term software component used by Lau and Wang [7]. In their interpretation of the term we would build applications by assembling components, either preexisting or built specifically for the application. Libraries as components, in the context of the SM can consist of a single class or multiple classes. The internal implementation is not significant. From the user of the library's point of view, only the interfaces provided are important.

Application A3, like Application A1, also has a required interface that uses Interface L1. We know from the enterprise level that System A also interfaces to System B through Interface B1. In Figure 4 we can see that System A delegates this responsibility to Application A3.

Application A4, while having no relationship with any of the other applications in System A, does have the responsibility of providing the interface with System E. We can see that System A has delegated this responsibility to Application A4 by the delegation connection to the external port Connection AE. This is an example of a direct connection. The nature of this relationship would be described in the architecture document interfaces section or in a separate interface design document.

One very important point to note is the relationship of System A the container to System A the component of the Enterprise X container. In the Enterprise X component diagram System A has a relationship to three other systems; two are interface relationships, one provided and one required, and the other is a direct connection. System A, the container, maintains those relationships as indicated by the external ports, Interface A, Interface B1 and Connection AE.

The formal software architecture document for this software would contain these diagrams as well as detailed textual descriptions of each component, interface and connection at each level. For the interfaces these would describe the nature of the interface such as data exchange or direct program-to-program communication. It might also include a reference to any standards that might apply. As for the components, specifically the system components of the enterprise level for example, the text description would provide information about which systems will be developed and which might be COTS products. Clearly not all information can be conveyed in just the component diagrams alone. However, it has been demonstrated that a great deal of information can. More importantly, the information that is provided is exactly the kind of information that might be overlooked had the design started without any consideration of software architecture.

Concluding Remarks

While the methodology described here can go a long way to improving our software engineering design drawings and documents, further work is needed to refine the methodology. More consideration needs to be given to the application layer and the macro and micro components layers of the original model. A formal method for the validation and verification of models created with this methodology is also needed. These offer only a few areas for further research.

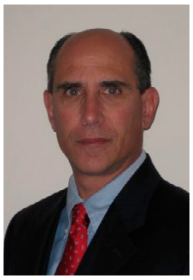
This paper has attempted to close, or at least reduce the width of, the gap between software architecture theory and practice. A methodology was described which demonstrates how to use UML component diagrams as a way to document and communicate software architectures clearly and in a reproducible fashion. This methodology leverages one of the two modern definitions of software architecture found on SEI's website and a lesser-known model for producing software architectures.

If software engineering is ever to achieve the same status as other engineering disciplines, or even approach that status, practitioners must be able to produce universally understood and reproducible design documents. The history of previous work in the area of software architecture has provided a rather stable theoretical foundation. UML, the de-facto standard for creating software engineering design diagrams provides the tools. It is up to us, the practitioners, to use these tools in the way they were intended. It is hoped that this paper demonstrates how to do just that.

Acknowledgements

The author wishes to thank Bob Nicholson, Steve Chappell, Mike Watts and Dr. Jiacun Wang for their review of this paper as well as their thoughtful comments.

ABOUT THE AUTHOR



Michael Tarullo has 29 years of experience in software design and development. Approximately half of this time was spent developing software in the mainframe environment. The remainder of his career has been spent designing and developing object-oriented software in C++, C#, and Java. He is currently an Enterprise Architect at L-3 Communications supporting System Wide Information Management, the FAA SOA initiative for the NextGen Air Traffic Management system. He also is an adjunct professor of Software Engineering at Monmouth University, where he teaches both undergraduate and graduate courses in Java Programming and Software Design. Mr. Tarullo has a Bachelors Degree in Geoscience from The New Jersey City University and a Masters Degree in Software Engineering from Monmouth University.

L-3 Communications
Contractor, FAA William J. Hughes
Technical Center
Atlantic City, NJ 08405
Phone : (609) 485-5294
E-mail: michael.ctr.tarullo@faa.gov
E-mail: michael.tarullo@l-3com.com

REFERENCES

1. <http://www.sei.cmu.edu/architecture/start/definitions.cfm>
2. Kruchten, P., Obbink, H. and Stafford, J.; The Past, Present and Future of Software Architecture; IEEE Software; March/April 2006
3. Shaw, M. and Garlan, D.; Software Architecture – Perspectives On An Emerging Discipline; Prentice Hall; 1996
4. Bass, L., Clements, P. C. and Kazman, R.; Software Architecture in Practice; Addison-Wesley; 2003, 2nd edition
5. Mowbray, T. J and Malveau, J.; CORBA Design Patterns; John Wiley & Sons; 1997
6. Tyree, J. and Akerman, A.; Architecture Decisions: Demystifying Architecture; IEEE Software; v.22, no.2, 2005
7. <http://bluehawk.monmouth.edu/~btepfenh/Courses/SE505/Sections/principlesdesign.html>
8. Booch, G., Rumbaugh, J. and Jacobson, I.; The Unified Modeling Language User Guide; Addison Wesley; 2nd Edition; 2005
9. Rumbaugh, J., Jacobson, I. and Booch, G.; The Unified Modeling Language Reference Manual; Addison Wesley; 2nd Edition; 2005
10. Booch, G., Maksimchuk, R. A., Engle, M. W., Young, B. J., Conallen, J. and Houston, K. A.; Object-Oriented Analysis And Design With Applications; Addison Wesley; 3rd Edition; 2007
11. Lau, K. and Wang, Z.; Software Component Models; IEEE Transactions On Software Engineering; October 2007, vol. 33, no. 10

Utilizing Design of Experiments to Reduce IT System Testing Cost

Kedar M. Phadke, Phadke Associates, Inc.
Madhav S. Phadke, Phadke Associates, Inc.

Abstract. Software and system testing cost the commercial and defense industry hundreds of millions of dollars annually. In addition, conducting each set of tests takes multiple man-months, delaying time to market of key technologies. In this current economic environment, organizations are looking for ways to reduce the cost of testing and time to market while ensuring that defects are not passed on to the customer. At the same time, organizations are very reluctant to change their standard testing processes due to the heavy cost of field failures, regulatory concerns, and risk-averse culture.

This paper describes a comparative study undertaken to assess the benefits of using Orthogonal Arrays (OA) for generating test plans in IT systems in the financial services industry. The formal process used for the comparative study consisted of enlisting the support of senior management and conducting multiple side-by-side pilots to compare the cost and risk of OA based testing versus the Business as Usual (BAU) testing practices. Our customers ran 20 side-by-side studies to evaluate the effectiveness of OA based testing and realized an average reduction in total test effort by 41%. In addition, all defects detected by the BAU process were detected by the OA based testing process. Further, in 40% of the cases, the OA based testing process found more defects. The cost and schedule savings translated to tens of millions of dollars in labor and schedule.

The paper also discusses the pros and cons of OA testing versus other testing approaches, namely, pairwise testing, N-Way testing, and classical Design of Experiments (DoE).

Utilizing OAs for system and software testing will significantly reduce cost, schedule and risk. For the aerospace and defense industries, OA testing will help address the current environment of tighter budgets and schedules while ensuring end users promised performance. This process is being adopted by several top tier defense and aerospace system developers for software and system testing and its applications have demonstrated significant reduction in both program cost and risk.

Overview of OA Testing Process

OAs are a mathematical tool that has been studied and utilized for centuries by mathematicians, scientists, and engineers for a variety of applications [1,2,3,4,5,6,11,12]. The most well known, Leonhard Euler, utilized OAs (also called Latin Squares) to cleverly arrange multiple ranks of military officers and for war games. One of the co-authors, Madhav Phadke, introduced the use of OAs for software testing while at AT&T Bell Laboratories in the 1980s, achieving great success for network and telecommunications system testing [7].

Consider a function to be tested with four parameters: A, B, C, and D. These parameters could be the arguments of the command line entered from the terminal, the state of an interface, input from a connecting device, or the initial states of internal parameters. Suppose each parameter has three possible levels as given in Table 1. This parameter-level table specifies the test domain consisting of 81 possible combinations of the test parameter levels. (In the Robust Design literature, "factor" is often used in place of "parameter.")

Test Parameter	Level 1	Level 2	Level 3
A	A_1	A_2	A_3
B	B_1	B_2	B_3
C	C_1	C_2	C_3
D	D_1	D_2	D_3

Table 1: Test Parameters and Levels

The job of a software tester is to attempt to break the system in every possible way so that all faults will be detected, which will therefore increase the likelihood of delivering fault-free software to the customer.

Table 2 shows the OA L9. It has nine rows and four columns. The rows correspond to test cases; the columns correspond to the test parameters. Thus, the first test case comprises Level 1 for each parameter, i.e., it represents the combination A1, B1, C1, D1. The second test case corresponds to the combination A1, B2, C2, D2, etc.

Test Number	Test Parameter A	Test Parameter B	Test Parameter C	Test Parameter D
1	1	1	1	1
2	1	2	2	2
3	1	3	3	3
4	2	1	2	3
5	2	2	3	1
6	2	3	1	2
7	3	1	3	2
8	3	2	1	3
9	3	3	2	1

An OA has the balancing property that, for each pair of columns, all parameter-level combinations occur an equal number of times. In OA L9, there are nine parameter-level combinations for each pair of columns, and each combination occurs once. Taguchi [8] and Madhav S. Phadke [9] provide a comprehensive discussion of OAs and their selection for specific applications. By conducting the nine tests indicated by L9, we can accomplish the following:

- ***Detect and isolate all single-mode faults.*** A single-mode fault is a consistent problem with any level of any single parameter. For example, if all cases of factor A at Level A1 cause error condition, it is a single-mode fault. In this example, tests 1, 2, and 3 will show errors. By analyzing the information about which tests show error, one can identify which factor level causes the fault. In this example, by noting that tests 1, 2, and 3 cause an error, one can isolate A1 as the source of the fault. Such an isolation of fault is important to fix the fault.
- ***Detect all double-mode faults.*** If there exists a consistent problem when specific levels of two parameters occur together, it is called a double-mode fault. Indeed, a double-mode fault is an indication of pairwise incompatibility or harmful interactions between two test parameters.
- ***Multimode faults.*** OAs of strength 2 can assure the detection of only the single- and double-mode faults. However, many multimode faults are also detected by these tests by virtue of the fact that OA-based tests are uniformly distributed in the test domain.

Real software testing problems tend to have dozens of parameters with two to 15 potential values per test parameter, thus manually determining appropriate OAs is a challenge for most software test professionals. Commercial tools for generating OAs for specific problems can be very helpful for this task. The cases studies in this paper were all conducted using a commercial software tool, rdExpert™ Test Suite, for OA generation [10].

Enterprise Mortgage IT System Validation

Several case studies have been conducted to validate the OA testing process for IT systems within the financial services industry. This section details one specific case study for a mortgage bank. The next section provides a summary of 20 similar studies conducted at 10 large financial services firms.

A major mortgage bank was revamping its enterprise IT system to better meet customer needs. The bank has several business processes geared towards different stages of mortgage processing. In the past, each business process had its own software systems, and the hand-over between processes were made manually. This caused delays in servicing customers and resulted in loss of business. For example, the bank had a system for accepting mortgage applications and a separate system for underwriting. This meant that once an application was accepted, it had to be manually input in the underwriting system, processed, and then a quote was manually input back into the application portal for the customer. In the new environment of customers demanding immediate feedback on mortgage applications, this was not fast enough. To address this customer need, the mortgage bank was developing an enterprise integration platform to automatically transmit data between the disparate systems and make the end-to-end process more efficient.

The bank had hired an outsourced provider to develop and test the integrated system. The scope of the system included all major business processes such as 1) loan application and approval, 2) payment acceptance and management, 3) data storage and access, and 4) corporate reporting and governance. At the time the bank began considering OA for testing, the outsourced team was seven months behind on delivery and 20% over budget.

The bank management culture was by tradition risk-averse so the teams were hesitant to change the current BAU testing approach. This risk-averse culture, coupled with the severe consequences of field failure and regulatory rules, made it even more difficult to change the process. To alleviate these concerns and at the same time assess the benefits of using OA for testing, the management decided to fund parallel teams to conduct testing of the application. One team would utilize the traditional BAU approach and the other team would utilize the OA testing approach. Both teams were tasked to complete the entire end-to-end process, including test case design, test scripting, test execution, defect analysis, and root cause identification. After completing the process, management would be able to evaluate the effects of using OA compared to BAU on cost, risk, and schedule.

The BAU process was an industry standard process and the key steps were as follows:

- Understand the most-likely customers.
- Understand the most-likely paths in processing customers' mortgage applications.
- Create several test scenarios that ensure all top-level requirements are covered, with majority focused on most-likely circumstances.
- Include crisis scenarios.

Table 3 provides a simplified view of the systems integration test planning scenarios. The highlighted values were considered most likely from a customer perspective.

Application	Loan Type	Credit Verification	Payment	Payment Amount	Customer Data Access	Datacenter Status
Online	Home Equity	Experian	Check through mail	Correct amount	Online reports	All online
Phone	Non-Traditional	Equifax	Bank transfer via phone	Underpayment	Mailed reports	1 Datacenter offline (Routine service)
Retail Center	Jumbo 1	Transunion	Debit card via phone	Overpayment	Online and mailed reports	2 Datacenters offiline (Critical)
Mail	Jumbo 2	Internal*	Bank transfer via online portal	Permature repayment (Termination)		
Partner Broker	Traditional		Debit card via online portal	Final payment (Termination)		
			Cash at retail center			
			Check at retail center			
			Third party transfer			

Table 3: Simplified View of Systems Integration Scenarios

Once the team had determined the most likely customer scenarios and paths, they would generate test cases to validate those specific situations. They would then add variations to the scenarios to include less likely customer scenarios, crisis situations, and other test cases suggested by experts. The team utilizing the BAU test process generated 188 test scenarios to validate the enterprise system.

The OA team utilized a different approach for selecting test scenarios and the key steps were as follows:

• Understand the requirements domain (determine the parameter-level table).
• Peer review the test parameters and levels.
• Generate test conditions based on OA.
• Prioritize the test plan for most likely and most important customer scenarios.

Instead of working on the most likely scenarios up front, the OA team first compiled a thorough summary of the key parameters and levels. Table 4 in the appendix provides an abridged summary of the test parameters and levels. Please note that some details have been changed to preserve client confidentiality. Prioritization of the test was completed at the end, just before execution. The OA team generated 81 test scenarios to validate the enterprise system. Both teams generated test scripts, executed the test plans, and evaluated defects. Table 5 provides a summary of the results.

The side-by-side comparison clearly shows that the OA approach detected all the unique defects detected by the BAU process. Thus, both methods have the same effectiveness. The OA approach reduced the test planning effort from 480 hours to 145 hours, a 70% reduction. The test execution effort is the sum total of the effort needed for the environment set up, running the tests, and defect analysis. The BAU approach required 1,670 hours for test execution while the OA approach needed only 890 hours, which represents a 47% saving. At the average loaded hourly cost of $72, the cost for the BAU approach was $154,800 whereas the cost of the OA approach was only $74,520, representing a 52% cost reduction.

Side-by-Side Studies at Multiple Financial Services Firms

Similar to the mortgage bank case study described above, 20 complete side-by-side studies were conducted at 10 large financial services firms in the U.S. and Europe.

Test Plan	No of Tests	Test Planning Effort (hrs)	Test Execution Effort (hrs)	Unique Defects Found
BAU (Business as Usual)	188	480	1670	12
OA (using rdExpert™ Software)	81	145	890	12
Savings Comparison	**107**	**335**	**780**	**Same Defect Coverage**

Table 5: Summary of Results: Enterprise Application Validation

Factor Name	Level 1	Level 2	Level 3	Level 4	Level 5	Level 6	Level 7	Level 8
Application	Online	Partner Broker	Phone	Mail	Retail Center			
Applicant History	New Customer	Previously Approved but No Loan Issued	Previously Denied	Previous Customer				
Loan Type	Jumbo 1	Traditional	Jumbo 2	Non-Traditional	Home Equity			
Credit Verification	Equifax	Experian	Internal*	Transunion				
Title Accreditation	Title Partner 2	Title Partner 3	Title Partner 1					
Appraisal Partner	Appraisal Partner 1	Not Necessary (Override)	Appraisal Partner 2					
Payment	Bank Transfer by Online Portal	Check through Mail	Debit Card by Online Portal	Cash at Retail Center	Check at Retail Center	Third Party Transfer	Debit Card by Phone	Bank Transfer by Phone
Payment Amount	Correct Amount	Premature Repayment (Termination)	Final Payment (Termination)	Overpayment	Underpayment			
Customer Data Access	Mailed Reports	Online and Mailed Reports	Online Reports					
Broker Comissions	Bank Transfer	Check through mail						
Datacenter Status	All Online	2 Datacenters Offline (Critical)	1 Datacenter Offline (routine service)					
Security Status	No Issues	Major Security Issues (Critical)	Small Security Issues					
Network Status	No Issues	Portions of Network Offline (routine service)						
Business Reports	Business Report 2	Business Report 1	Tax/Accounting Report 1	Business Report 4	Tax/Accounting Report 2	Business Report 3		
Governance	Independent Private Entity	Government Entity						

Table 4: Abridged List of Test Parameters and Values for Enterprise System Validation

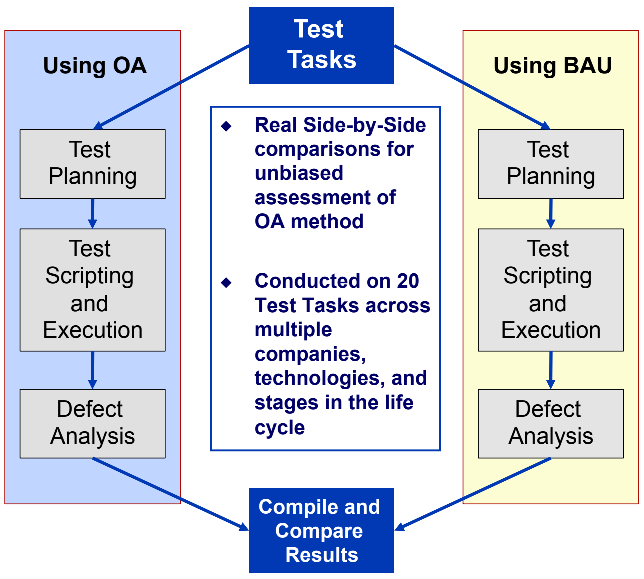
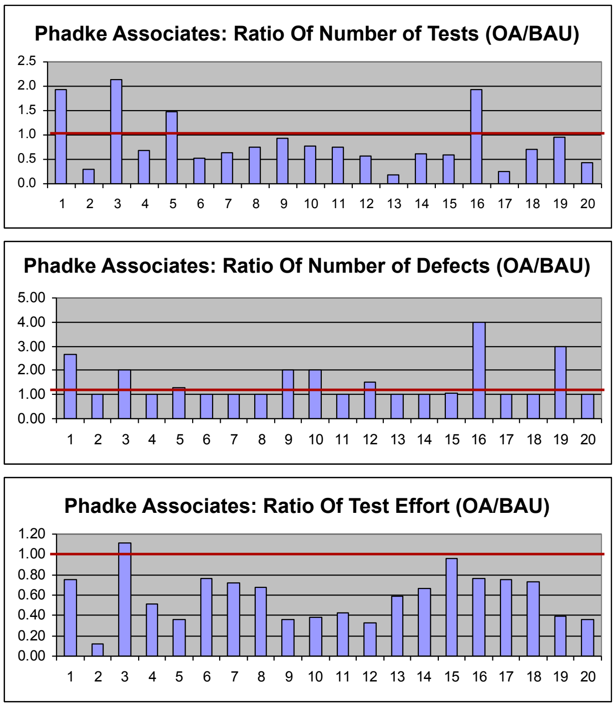

Test Tasks
Using OA
Test Planning
Test Scripting and Execution
Defect Analysis
Real Side-by-Side comparisons for unbiased assessment of OA method
Conducted on 20 Test Tasks across multiple companies, technologies, and stages in the life cycle
Using BAU
Test Planning
Test Scripting and Execution
Defect Analysis
Compile and Compare Results

Figure 1: Side-by-Side Process

Four of the firms were investment banks, three were health/auto/property/liability insurance companies, one was a life insurance company, and two were retail/mortgage banks. All of the case studies included parallel teams so that management could get truly unbiased information to compare their BAU process versus the OA testing process. Figure 1 shows a flow diagram for the side-by-side process utilized by the companies. Note that the process is the end-to-end testing process including test planning, scripting, execution, and defect analysis.



Graph 1: Result of Side-by-Side Studies

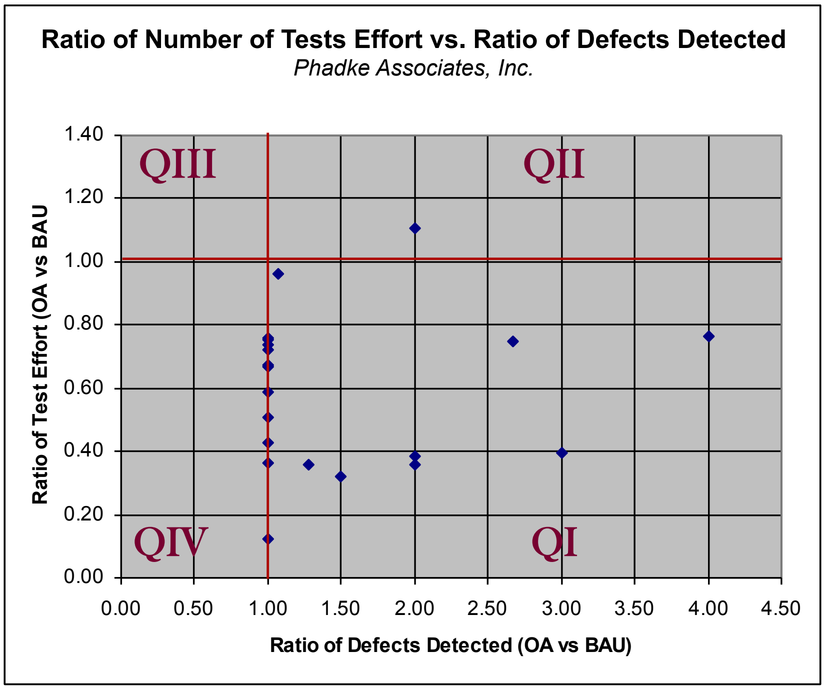
The objective of the side-by-side studies was to ascertain that, 1) OA tests do not increase risk (do not miss defects versus BAU), and 2) OA testing process does not increase total test effort. Graph 1 displays the ratios of test cases, defects detected, and total test effort for all 20 case studies.

The key findings for the management teams were that:

• All unique defects found by BAU were detected by OA testing process in each of the 20 cases.
• In 40% of the cases, OA tests detected more defects. Thus, in many cases, OA tests provided more risk reduction than BAU.
• Total test effort was reduced by 41% on average. This was a saving of tens of millions of dollars.
• In the four cases where the number of test cases or test effort increased, more defects were detected, so time was utilized productively.

In addition, the ratio of test effort (OA/BAU) is graphed versus the ratio of defects detected (OA/BAU) to further compare the effectiveness of OA testing versus BAU at the 10 Financial Institutions. Graph 2 displays the ratio of effort versus the ratio of defects detected. There are eight points in Quadrant I. These points represent the cases where the test effort of OA is less than or equal to BAU and the number of defects is greater than the number detected by BAU. This is the most desired quadrant to be in. Eleven points are the line bordering Quadrants I and IV. These points represent cases where both OA and BAU find the same number of faults. However, for these points the test effort for OA is less than the test effort for BAU. One point lies in Quadrant II. For this case, the OA required more test effort and OA found more defects. In other words, there was more test effort but the team found more defects. There are no test points in Quadrant III and Quadrant IV that are the least desirable quadrants to be in.

Thus the side-by-side case studies clearly demonstrated that the OA testing process is effective for significantly reducing the test effort and also simultaneously reducing the risk for IT testing in the financial services industry.



Graph 2: Ratio of Test Effort vs. Ratio of Defects

Command and Control System

Similar to the financial services industry, the defense and aerospace industries are also facing significant pressure to reduce program cost and schedule. The OA testing process has been successfully applied by several teams in both government and industry. One of the more pressing challenges is effective testing of complex software intensive systems. The Defense Information Standards Agency (DISA) conducted a retrospective pilot of the OA testing process for one of their key modules of a software-based command and control system. The module was designed to retrieve and process data from multiple data sources and display the data in a composite picture. The data sources included human intelligence, measurement and signature intelligence, signals intelligence, Blue/Red Force data, friendly and hostile data from air, ground and sea, and several other sources. The contractor had developed a test plan to validate the module utilizing their BAU practices. Utilizing the OA testing process with the assistance of Phadke Associates, DISA was able to reduce the test plan size by more than 50% and estimated that the test planning effort could be reduced from 24 staff weeks to one staff week. The estimated savings of the reduced test plan and staff savings was $377,000. In addition, the analysis demonstrated that the original contractor test plan had over 340 test gaps and all gaps were eliminated by the more efficient OA testing process. This retrospective pilot demonstrated significant capability of the OA testing process to reduce test cost, risk, and schedule for defense software systems.

Comparison With Other Test Planning Methods

During the piloting, the teams also compared OA testing with other common test planning methods, namely pairwise, N-Way, and classical DoE. Based on their findings they decided to conduct side-by-side testing pilots using only the OA testing process. Table 6 lists a summary of pros and cons identified for each method versus OA testing.

The teams realized that pairwise testing, a method to ensure that each pair is tested at least once, had the potential to reduce the number of test cases versus OA in some instances; however, the additional cost of defect analysis outweighed the potential reduction in execution cost. Since the test cases created by the pairwise method can be unbalanced, it requires significantly more time to isolate the root causes of defects. In addition, they found it challenging to effectively assess performance in terms of the statistical properties like mean and variance. In fact, to effectively conduct analysis of faults and results, the teams found they had to run several additional test cases. OAs distribute test cases uniformly in the multidimensional test domain [11], whereas pairwise test cases tend to be sparser in some regions than other regions. Consequently, pairwise tests can have less ability to detect faults compared to OAs.

N-Way is a test planning method that ensures that each N-Way combination of parameters is tested at least once. For example, users could specify all three-way combinations, in which case each triplet would be tested at least once in the test plan. One of the first challenges the teams discovered when examining the N-Way testing method was that the number of test cases, even for triplets, was significantly larger than their current BAU process, thus implying a significantly increased test execution cost. Also, similar to pairwise testing, the N-Way test cases are unbalanced and require significantly more time and effort to conduct defect analysis and assess performance. Due to these cost and schedule increases, this method was deemed financially prohibitive. Proponents of N-Way testing site the need to identify rare high order defects, such as five-way or six-way defects. Upon deeper analysis, teams have realized that the preferred approach to address this need is to use hierarchical test plans based on utilizing the broad knowledge of the system (or system-of-systems) architecture. This approach is more effective for detecting high order defects and also significantly more economical compared to the five-way or six-way test plan.

Test Method	Pros	Cons
Business as Usual	Can be effective and efficient with highly skilled gurus and lots of time. (These are both rare commodities!)	Test plan effectiveness highly dependant on the individual. No consistency across organizations.
Pairwise (not OA)	Sometimes fewer tests than OA	Unbalanced test cases so debugging is challenging and performance assessments for continuous outputs even more challenging. Costs for test data analysis are much larger.
N-Way (Greater than 2-way)	Generally better coverage than OA and Pairwise	Significantly more tests so not affordable in today's economic environment. The cost is almost always much more than Business as Usual. Tests are unbalanced so same debugging and performance assessment challenges as Pairwise.
Classical Design of Experiments	Geared towards statistical modeling	Requires significant amount of staff training and expert guidance. Very difficult to cost effectively implement on a broad scale. Doesn't effectively address the multi-level designs that are necessary for system and software testing.

Table 6: Comparing other Test Planning Methods vs. OA testing

Classical DoE is primarily aimed at model building that is not the objective for a majority of testing tasks, especially when it comes to software and IT testing. The methodology often emphasizes Resolution IV designs with repetitions that result in significantly more tests, thus making the method financially unaffordable, similar to N-Way testing. To combat the financial concerns, practitioners of classical DoE often recommend reducing the number of factors or restricting the number of levels of each factor; however, this technique increases the risk of missing faults and adds significantly more to the downstream program cost and risk. This is particularly challenging for software and IT testing problems that involve mixed level designs with numerous factors having more than two levels (often many more levels). For example, if you have five data types for a particular test parameter, you will have to restrict your test to only two of those data types. Another key challenge is the classical DoE concept of repetitions that are necessary for building confidence in the statistical models. For software and IT systems, repetitions add significant cost but very little additional technical information.

Conclusion

The advantage of utilizing OAs for testing was demonstrated through 20 real end-to-end case studies where the OA process was run in parallel with the BAU process for IT testing at 10 large financial services institutions. OA-based testing resulted in a 41% reduction in total test effort (labor hours) and in all 20 cases, all defects detected by the BAU process were detected by the OA process. In 40% of the cases, the OA based testing process found more defects. The cost and schedule savings for these cases translated to tens of millions of dollars in labor and schedule.

The technical and managerial challenges for software and system testing in the defense and aerospace industry parallel those in the financial services industry in both scale and pressing need for "defect free" system delivery. Similar to the financial services industry, several defense and aerospace companies have piloted and are adopting the OA testing process and the rdExpert Test Suite Software. The results so far show that OA testing will help defense and aerospace industries meet the current challenge of tighter budgets and schedules while confidently delivering the end users promised performance.

Disclaimer

ABOUT THE AUTHORS

Kedar M. Phadke is Vice President of Phadke Associates, a global consultancy and software company specializing in statistical tools for improving testing and design productivity. Kedar has led numerous deployments for improving test and design effectiveness. He has a MS in Statistics, MS in Management, and a BS in Economics from the Wharton School, University of Pennsylvania.

Kedar M. Phadke
Vice President
Phadke Associates, Inc.
1 Shawnee Court
Colts Neck, NJ 07722
E-mail: kedar@phadkeassociates.com

Dr. Madhav S. Phadke is the Founder and President of Phadke Associates, Inc. He is an ASQ Fellow and the author of the first engineering textbook on Robust Design Methods in the U.S., "Quality Engineering Using Robust Design". He holds a Ph.D. in Mechanical Engineering and MS in Statistics from the University of Wisconsin - Madison, MS in Aerospace Engineering from the University of Rochester, and a BTech in Mechanical Engineering from the Indian Institute of Technology - Mumbai. Prior to founding Phadke Associates, Dr. Phadke was a manager in AT&T Bell Labs, a visiting scientist at the IBM Watson Research Center, and a Research Associate at the Army Math Research Center.

Madhav S. Phadke, Ph.D.
President
Phadke Associates, Inc
1 Shawnee Court
Colts Neck, NJ 07722
E-mail: madhav@phadkeassociates.com

REFERENCES

1. Addelman, S., "Orthogonal Main Effect Plans for Asymmetrical Factorial Experiments," Technometrics, Vol. 4, 1962, pp. 21-46.
2. Kempthorne, O., The Design and Analysis of Experiments, Robert E. Krieger Publishing, New York, 1979.
3. Plackett, R.L. and J.P. Burman, "The Design of Optimal Multifunctional Experiments," Biometrika, Vol. 33, pp. 305-325.
4. Seiden, E., "On the Problem of Construction of Orthogonal Arrays," Annals of Mathematical Statistics, Vol. 25, 1954, pp. 151-156.
5. Rao, C.R., "Factorial Experiments Derivable from Combinatorial Arrangements of Arrays," Journal of Royal Statistical Society, Series B, Vol. 9, 1947, pp. 128-139.
6. Raghavrao, D., Construction of Combinatorial Problems in Design Experiments, John Wiley and Sons, New York, 1971.
7. Brownlie, Robert, James Prowse, and Madhav S. Phadke, "Robust Testing of AT&T PMX/StarMAIL Using OATS," AT&T Technical Journal, Vol. 71. No. 3, May/June 1992, pp. 41- 47.
8. Taguchi, Genichi, System of Experimental Design, Don Clausing, ed., UNIPUB/Kraus International Publications, New York, Vols.1, 2, 1987.
9. Phadke, Madhav S., Quality Engineering Using Robust Design, Prentice Hall, Englewood Cliffs, N.J., 1989.
10. rdExpert™ Test Suite Software. Published and distributed by Phadke Associates, Inc. <http://www.phadkeassociates.com>.
11. Phadke, Madhav S., "Planning Efficient Software Tests" Crosstalk: Journal of Defense Software Engineering, Published by the Software Technology Support Center, October 1997.
12. Phadke, Madhav S., "Robust Testing: A Process for Efficient Fault Detection and Isolation", Aerospace Testing Seminar, 2006.

Defect Management Using Depth of Inspection and the Inspection Performance Metric

T.R. Gopalakrishnan Nair, Aramco Endowed Chair-Technology, PMU, KSA
V. Suma, RIIC, Dayananda Sagar Institutions

Abstract. Advancement in fundamental engineering aspects of software development enables IT enterprises to develop a more cost effective and better quality product through aptly organized defect management strategies. Inspection continues to be the most effective and efficient technique of defect management. To have an appropriate measurement of the inspection process, the process metric, Depth of Inspection (DI) and the people metric, Inspection Performance Metric (IPM) are introduced. The introduction of these pair of metrics can yield valuable information from a company in relation to the inspection process.

Introduction

A defect in software is expensive especially when it dwells and manifests. One of the prevailing challenges in the software industry is therefore the production of defect-free software [1]. The continuance of IT enterprise, hence, depends upon the choice of apt defect management strategies in order to generate defect-free software.

Quality control and quality assurance techniques are the two most successful defect management strategies. Quality control activity is for the product and quality assurance techniques are for the process. The current trends in the industry concentrate on testing, which is a quality control activity. However, what matters is the process through which a product is developed, and therefore excellence in the process plays a vital role towards delivery of high quality products. Among several techniques applied for quality assurance in the software field, like walkthrough, inspection, and review, inspection is one of the promising techniques for defect management. Despite its perceptible significance, inspection is either very casually treated or more often overlooked and many times it is maintained only for accounting purposes. One of the rationales being projected to escape the vital process step is identifying it as a mind-numbing, lengthy activity rather than a quality improvement process.

Since quality is a quantifiable unit, this article aims to draw the attention of the software community including management, developing teams, stakeholders, and outsourcing agents to an important aspect of bringing in a cultural change. It is worth for the industry to notice and comprehend the connotation of software inspection as process integration introduced recently by the implementation of a pair of metrics that are meant to measure the quality level of inspection process and further measure the competency of the people. The two metrics are DI, a process metric, and IPM, a people metric [2].

The DI Metric

Let us have a closer look at the apparently simple but powerful concept of DI. Let N_i be the number of defects captured by the inspection process and T_d be the number of defects captured by both inspection and testing approaches.

$$DI = N_i / T_d$$

Equation A:

DI can be measured phase-wise or before the deployment of the product using the above metric.

DI evaluation is realized in two phases. In the first phase, DI is calculated using shop floor defect count for a particular set of projects. This phase enables the software company to analyze the depth in which inspection process has occurred for a set of particular projects either phase-wise or at the project level. From our deep and rigorous investigations carried out across several service-based and product-based software industries of various production capabilities, it is found that the DI value varies from project to project. The metric is distinctive as it quantifies the inspection process with measurable levels, which is not observed in current industry standards. DI is considered to be in the range of zero to one where zero is nil performance and one indicates 100% defects captured exclusively through inspection process, which is hardly ever possible. An inspection level of 0.3 to 0.5 is considered normal inspection process and a level of 0.5 onwards requires high competency in the process [2].

Table 1 specifies the ranges of DI values [2]. With this chart, it is now possible for software personnel and all stakeholders to identify the maturity level of the company and enable them to either continue with the existing level or formulate strategies towards up gradation of their level. An additional strength of this mode of quality measurement is to throw light towards prediction of desired level of inspection.

DI	Inspection performance
0 - 0.1	Worse (W)
0.1 – 0.2	Very Low (VL)
0.2 – 0.3	Low (L)
0.3 – 0.4	Normal (N)
0.4 – 0.5	Above Normal (AN)
0.5 – 0.6	High (H)
0.6 – 0.7	Very High (AV)
0.7 – 0.8	Best (B)
0.8 – 0.9	Excellent (E)
0.9 - 1	Ideal (I)

Table 1: Range of DI Values

Prediction of quality of inspection process is not yet achieved. DI is a process metric that can predict the quality of software inspection process. The second phase consists of prediction of DI value for a new project through the approach of mathematical modeling scheme, which uses Multiple Linear Regression (MLR) models.

Prediction of DI is through the evaluation of process coefficients from the historical projects. Process coefficients are a set of β constants (β_0 to β_4) which are evaluated using least square estimates or using Matlab support. A minimum of five projects (P_1 to P_5) is required to evaluate the process coefficients. However, at a larger scale, depending on the history of the company and the past records of the projects that the company had handled, several groups of samples can be taken. It is also observed from the investigation made by several researches that effectiveness of inspection is influenced by four major and mutually exclusive parameters [2]. They are:

x_1 = inspection time
x_2 = preparation time
x_3= number of inspectors
x_4= experience level of inspectors

Having obtained the process coefficients and substituting desirable values to the inspection influencing parameters, it is now possible for the software company to predict the value of dependent parameter Y (DI) as given in equation (B) [3].

$$Y = \beta_0+\beta_1 x_1+ \beta_2 x_2+ \beta_3 x_3+ \beta_4 x_4 + e$$

Equation B:

Matrix representation for the prediction of DI is given in equation (C).

Software companies using the stabilized process coefficients can now predict the desired level of DI for any project (P_i) by modulating the inspection influencing parameters. Alternatively, with the evaluation of process coefficients it is also possible to tune the values of the inspection influencing parameters to achieve the desirable DI value. Table 2 illustrates the DI computation of 15 projects that are sampled from various product-based and service-based software industries. The sampled projects depicted throughout this article are similar types of projects that are developed using Java and operate in a similar type of environment.

Discernible benefits of DI in software organization are:

1. DI is a quality metric introduced in order to quantify the depth in which the inspection process is performed.
2. The objective of introducing DI as a defect detection metric is to enable one to analyze the defect capturing ability of the company through an inspection approach.
3. DI is a defect preventative metric whose implementation in the software industry acts as a lesson learned from previous projects with regard to the depth in which inspection is conducted and thus indicates the inspection team either "to improve the inspection performance" or "to maintain the desired level of inspection."
4. The aim of DI as an indicator metric is to inform the test team of the depth in which defects are detected through the

$$[DI] = [X][\beta] + [E] \text{ where}$$

$$[DI] = [Parameters] \times [processCoefficients] + [\text{Error Term}]$$

$$DI = \begin{bmatrix} DI_1 \\ DI_2 \\ DI_3 \\ DI_4 \\ - \\ - \\ DI_n \end{bmatrix} \quad \beta = \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \beta_3 \\ \beta_4 \end{bmatrix} \quad X = \begin{bmatrix} 1 & x_{11} & x_{12} & x_{13} & x_{14} \\ 1 & x_{21} & x_{22} & x_{23} & x_{24} \\ 1 & x_{31} & x_{32} & x_{33} & x_{34} \\ 1 & x_{41} & x_{42} & x_{43} & x_{44} \\ - & - & - & - & - \\ - & - & - & - & - \\ 1 & x_{n1} & x_{n2} & x_{n3} & x_{n4} \end{bmatrix} \quad E = \begin{bmatrix} e_1 \\ e_2 \\ e_3 \\ e_4 \\ - \\ - \\ e_n \end{bmatrix}$$

Equation C:

	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12	P13	P14	P15
Project hours(*)	250	263	300	507	869	1806	2110	4248	4586	4644	6944	7087	7416	8940	9220
DI at req. phase	0.53	0.49	0.67	0.52	0.33	0.48	0.5	0.46	0.39	0.27	0.44	0.44	0.49	0.44	0.47
DI at des. phase	0.5	0.38	0.46	0.54	0.42	0.5	0.44	0.49	0.44	0.4	0.64	0.46	0.57	0.45	0.43
DI at imp. phase	0.5	0.57	0.44	0.53	0.37	0.21	0.39	0.51	0.51	0.4	0.55	0.5	0.46	0.47	0.49
Avg DI	0.51	0.48	0.52	0.53	0.37	0.40	0.44	0.49	0.45	0.36	0.54	0.47	0.51	0.45	0.46
Tc (%)	96.0	95.0	91.5	96.0	89.8	87.0	92.0	95.4	96.5	88.3	96.9	96.5	93.1	95.8	92.3

(*) Total project time is measured in terms of person hours and contains documentation times, training time and release time etc., which are not relevant for this discussion; P = Project; req – Requirements analysis phase; des – Design phase; imp – Implementation phase; Avg - Average; Tc –Total defects captured in the complete project

Table 2: DI Estimation

inspection approach. Thereby, it directs the test team to frame effective strategies to eliminate remaining defects.

5. The rationale for the introduction of DI is to provide a deep visibility to the inspection team, the company management, outsourcing agents, and other stakeholders about the depth in which inspection is performed and thereby to provide enough transparency in the process.

6. It is observed that CMMI® Level 4 and above certified software industries are capable of performing inspection with an average DI value of 0.4 to 0.5. This demands that the test team puts in a certain amount of test effort to detect and eliminate remaining defects. A decrease in DI value further demands increased test effort, test time, increased developmental cost, and rework cost.

7. Project success depends upon quality [4]. Business success depends upon cost of quality [5]. According to the cost quality analysis, cost of rework is usually several orders higher at final stages when compared to quality implementation at initial stages [6]. The variation depends on the phase in which a defect originated and was later detected. The philosophy and culture that we propagate here offers a watertight control over defect management with appropriate quantitative metrics and methods in the process domain. Hence, the DI metric, with its distinguishing feature of giving process visibility, paves the way for stakeholders to control their developmental cost.

8. Implementation of DI is therefore a billion dollar savings to a software company since they are now able to visualize the depth in which all static defects are recovered and hence plan only towards detection and removal of dynamic defects through testing activities.

9. The existence of a DI metric is an eye opener for all stakeholders including clients and outsourcing agents to justify and control the developmental cost. With this metric, they are therefore in a position to substantiate the quality of the product and the maturity level of the company dynamically from project to project and predict a price tag for their project.

10. The DI metric brings out the variation in quality level of functional inspection and predicted inspection. This knowledge further enables the developing team to equip themselves towards their augmentation activities in order to endure in the competitive atmosphere of software industry.

The IPM Metric

Effectiveness of the inspection process depends on the people who drive the process. However, there are no software quality metrics existing to measure the performance of the inspection team within the constraints of major inspection affecting parameters, namely 1) inspection time, 2) inspection preparation time, 3) number of inspectors, 4) experience level of inspectors, and 5) complexity of the project that is measured in terms of function points [2]. The IPM metric helps a software company to make decisions toward the selection of appropriate values to the aforementioned parameters subsequently opting for the desirable team performance.

$$\mathrm{IPM} = \mathrm{N_i} / \mathrm{IE}$$
$$\text{where } \mathrm{IE} = \mathrm{N} \times \mathrm{T}$$
$$\text{and } \mathrm{T} = \mathrm{I_t} + \mathrm{P_t}$$

Equation D:

Let N_i = Number of defects captured by inspection process and IE = Inspection Effort

Where IE = Total number of inspectors (N) × Total amount of inspection time (T)

Total amount of inspection time (T) = Actual inspection time (I_t) + Preparation time (P_t) , (taken per person)

Where IE = Total number of inspectors (N) × Total amount of inspection time (T)

Total amount of inspection time (T) = Actual inspection time (I_t) + Preparation time (P_t)

IPM can be realized in two stages. In the first stage, number of defects captured by the inspection team within the aforementioned parameter constraints for any particular project is found using shop floor defect count. This mode of IPM calculation enables the software team to measure the team performance properties.

The second stage of realizing IPM is to predict IPM value for a new project using a mathematical scheme. Prediction of IPM for a project is realized using MLR models.

Let (β_0 to β_5) = team coefficients,
x_1 = inspection time
x_2 = preparation time
x_3= number of inspectors
x_4= experience level of inspectors
x_5 = the complexity of the project measured using function point analysis in a logarithmic scale.

$$Y = \beta_0+\beta_1 x_1+ \beta_2 x_2+ \beta_3 x_3+ \beta_4 x_4 + \beta_5 x_5 + e$$

Equation E:

Thus, with the system of MLR equations, a set of team coef¬ficients is evaluated using: Equation F.

Evaluation of β coefficients is realized using Least Square Technique using Matlab support and requires a minimum of six empirical projects for the evaluation purpose [7]. Let Y represent IPM value for a project that can be obtained by substituting the parameter values and team coefficients in equation (E) as shown below.

Thus, with the system of MLR equations, a set of team coefficients is evaluated using:

Thus, having stabilized the team coefficients, the manager can obtain the desired IPM by appropriately tuning the inspection influencing parameters for the given complexity of the project [7]. Table 3 illustrates the computed IPM values for the previously sampled 15 projects.

$$[IPM] = [X][\beta] + [E] \text{ where}$$

$$[IPM] = [Parameters] \times [TeamCoefficients] + [\text{Error Term}]$$

$$M = \begin{bmatrix} IPM_1 \\ IPM_2 \\ IPM_3 \\ IPM_4 \\ - \\ - \\ IPM_n \end{bmatrix} \quad \beta = \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \beta_3 \\ \beta_4 \\ \beta_5 \end{bmatrix} \quad X = \begin{bmatrix} 1 & x_{11} & x_{12} & x_{13} & x_{14} & x_{15} \\ 1 & x_{21} & x_{22} & x_{23} & x_{24} & x_{25} \\ 1 & x_{31} & x_{32} & x_{33} & x_{34} & x_{35} \\ 1 & x_{41} & x_{42} & x_{43} & x_{34} & x_{45} \\ - & - & - & - & & - \\ - & - & - & - & & - \\ 1 & x_{n1} & x_{n2} & x_{n3} & x_{n4} & x_{n5} \end{bmatrix} \quad E = \begin{bmatrix} e_1 \\ e_2 \\ e_3 \\ e_4 \\ - \\ - \\ e_n \end{bmatrix}$$

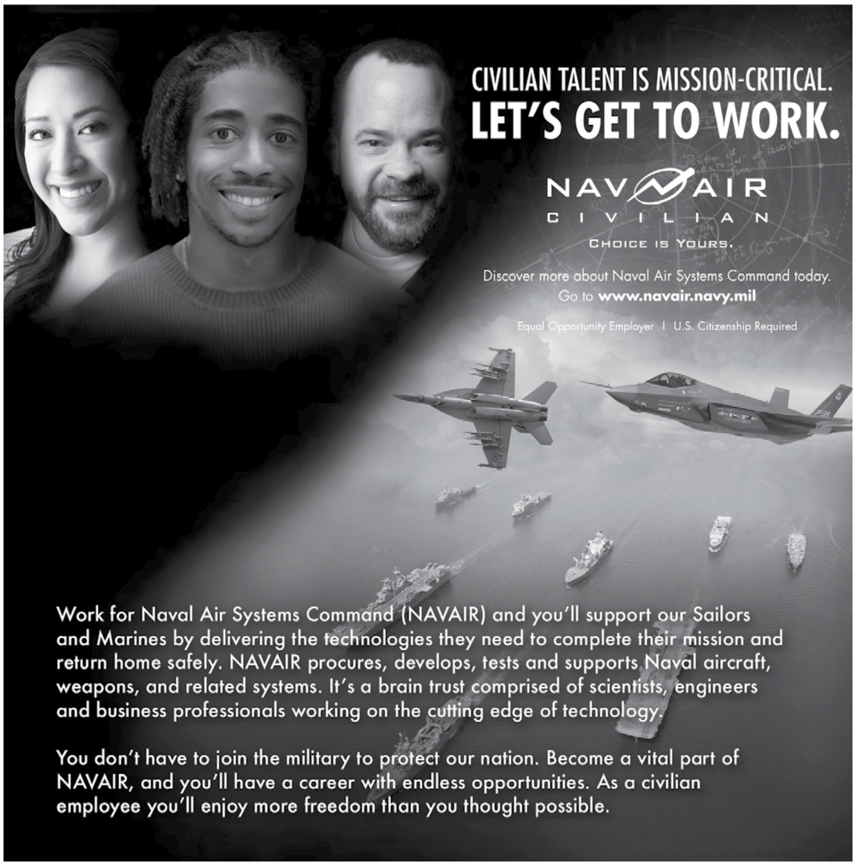
Equation F:

	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12	P13	P14	P15
Project hours(*)	250	263	300	507	869	1806	2110	4248	4586	4644	6944	7087	7416	8940	9220
IPM at req. phase	4.57	5.40	6.89	5.71	2.714	3.11	1.25	0.66	0.36	1.03	0.35	0.66	0.51	0.73	0.60
IPM at des. phase	0.71	0.67	1.00	1.17	0.889	0.86	0.44	0.20	0.22	1.56	0.33	0.30	0.24	0.29	0.17
IPM at imp. phase	0.36	0.70	0.37	0.36	0.412	0.18	0.13	0.20	0.25	0.32	0.20	0.50	0.20	0.24	0.12
Avg IPM	1.88	2.25	2.75	2.41	1.34	1.39	0.61	0.35	0.27	0.97	0.29	0.49	0.32	0.42	0.29
Tc (%)	96.0	95.0	91.5	96.0	89.8	87.0	92.0	95.4	96.5	88.3	96.9	96.5	93.1	95.8	92.3

Table 3: IPM Estimation

Observable gains of IPM are:

1. IPM is introduced as an effort analysis metric in order to uniquely identify the effort put forth by the team for inspection.
2. The objective of introducing IPM as a quality indicator is to indicate the level of quality being achieved by the inspection team.
3. The main purpose of IPM is to provide transparency and visibility to the customers and thereby help them to justify and control the developmental cost.
4. Introduction of IPM in the software development cycle enables the inspection team to evaluate their performance level.
5. With the implementation of IPM as a software quality metric in the inspection process, the managers of the software company can now choose the team specification in order to achieve desired inspection effort.
6. The goal of IPM is to provide a deep visibility to the inspection performance for stakeholders, clients, managers, and outsourcing agents.
7. Existence of IPM in the software industry enables management to dynamically justify and control the staff cost to every project based on team performance.
8. IPM further acts as an awareness metric for the inspection-performing team to be aware of the team's performance and to appropriately formulate strategies towards their improvement activities.
9. Implementation of IPM therefore acts as a metric to save the economy of the company as it provides deep visibility of the team's performance in effective defect capturing abilities.
10. IPM further encourages the test team to train themselves for the capturing of residual defects.


CIVILIAN TALENT IS MISSION-CRITICAL.
LET'S GET TO WORK.
NAVAIR
CIVILIAN
CHOICE IS YOURS.
Discover more about Naval Air Systems Command today.
Go to www.navair.navy.mil
Equal Opportunity Employer | U.S. Citizenship Required

Work for Naval Air Systems Command (NAVAIR) and you'll support our Sailors and Marines by delivering the technologies they need to complete their mission and return home safely. NAVAIR procures, develops, tests and supports Naval aircraft, weapons, and related systems. It's a brain trust comprised of scientists, engineers and business professionals working on the cutting edge of technology.

You don't have to join the military to protect our nation. Become a vital part of NAVAIR, and you'll have a career with endless opportunities. As a civilian employee you'll enjoy more freedom than you thought possible.

DI and IPM are applicable across a variety of projects with the limitation that the ranges of values for acceptable performance differ from one type of project to another like innovative projects, legacy projects, etc.

Conclusion

Inspection is one of the most promising techniques of defect management that enhances industrial productivity and quality. Inspection is a challenging task that provides a platform for a professional inspector to exhibit his competency to detect a maximum number of defects under time, cost, and resource constraints. It is essential for quality managers to apply appropriate metrics to monitor the effectiveness of inspection and the performance level of inspection. They can make use of two newly introduced metrics, DI and IPM, with their desirable band of operation to judge the level of success of process.

The investment of implementing DI and IPM over the existing process certainly demands a cost. However, it is inline with the dictum, "It is not just the investment that matters for quality, but also the right kind of investment." The implementation of DI and IPM is a right kind of investment that improves the position of a company's value to the market and stakeholders. The process metric DI and people metric IPM could be effectively used by clients, sponsors, and users to judge the perfection of developed, highly qualified software. Further, these metrics will pave the way for justifying the developmental cost with deep visibility into the process.

Due to the value of DI and IPM, the required effort to capture a maximum amount of defects is reduced. Managers get the added advantage of monitoring team performance, project after project, in a convincing way using numerical estimations through characteristic coefficients of the team or the company.

The DI and IPM value can now be either estimated based on defect counts from the shop floor or they can be predicted through the process coefficients and team coefficients that were empirically evaluated using a large sample of projects. Once the coefficients are stabilized, it is possible to predict the achievable DI and IPM through our model, without depending on the defect count. It implies that the managers can have the ability to finalize the inspection influencing parameters while planning the inspection process to achieve a particular DI. Having finalized the IPM that a company should achieve, it can tune the number of persons doing inspection, the experience of each person, and the time to be spent by each person to achieve the desired quality level of IPM.

Since DI and IPM are directly affecting defect management, development of a 99% defect-free product is possible by choosing appropriate values of parameters influencing DI and IPM. In order to realize the effectiveness of DI and IPM, it is absolutely necessary for quality-conscious outsourcing agencies and companies to run a piloted rollout of the inspection strategy.

DI and IPM (Nair-Suma metric) are valid across a spectrum of projects. But, it is important to note the ranges of values for acceptable performance differs from one type of project to another.

Acknowledgment:

The authors would like to acknowledge all the industry personnel who supported this work directly and indirectly within the framework of nondisclosure and thank all those who supported us to carry out the research successfully. Authors would further like to acknowledge the CrossTalk Editorial Board and the reviewers for their valuable suggestions.

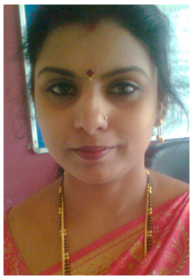
Disclaimer:

CMMI® is registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

ABOUT THE AUTHORS

Dr. T.R. Gopalakrishnan Nair was the Vice President for Research in DS Institutions, Bangalore, India and currently holds the Saudi Aramco Endowed Chair of Technology in PMU, KSA. He worked in Indian Space Research and Indian Industries and has published extensively in computer field. Dr. Nair, a senior member of IEEE and ACM, was given the national technology award "PARAM" in 1992. He is interested in Software Engineering, Advanced Networking, Real-time Systems and Bio Computing.

T.R. Gopalakrishnan Nair
Saudi Aramco Endowed
Chair - Technology
Computer Engineering and Science
Prince Mohammad Bin Fahd University
PO Box 1664, Alkhobar 31952
Phone: +966 896 4554
Fax: + 91 896 4566
E-mail: trgnair@ieee.org
E-mail: trgnair@gmail.com



Dr. V. Suma is Head-Advanced Software Engineering Research Group of Research and Industry Centre and Professor in Dayananda Sagar Institutions, India. With a Ph.D. in Computer Science and Engineering, she is associated with several leading software industries, universities. Being an IEEE member, she has authored international book chapter and several research papers published at reputed International Journals, National, International conferences. Her main areas of interest are Software Engineering, Database Management System and Information Systems.

Suma. V.
Head, Advanced Software Engineering
Research Group,
Research and Industry
Incubation Center,
Professor, Department of Information
Science and Engineering
Dayananda Sagar Institutions,
Kumaraswamy Layout,
Bangalore-560078, India
Phone: +91 9448305148
Fax: +91 80 23637053
E-mail: Sumavdsce@gmail.com

REFERENCES

1. Humphrey, Watts S. "The Software Quality Challenge."CROSSTALK: The journal of Defense Software Engineering (June 2008).
2. Nair, Gopalakrishnan T.R., Suma V. "Impact Analysis of Inspection Process for Effective Defect Management in Software Development." Software Quality Professional Journal, American Society for Quality (ASQ), (March 2010) 4-14.
3. Nair, Gopalakrishnan T.R., Suma V. "A Paradigm for Metric Based Inspection Process for Enhancing Defect Management". ACM SIGSOFT, (May 2010).
4. Gilb, Tom. "The 10 Most Powerful Principles for Quality in Software and Software Organizations. " CROSSTALK: The journal of Defense Software Engineering (November 2002).
5. Krasner, Herb. "Using the Cost of Quality Approach for Software." CROSSTALK: The journal of Defense Software Engineering (March 2006).
6. Khaled El Emam. "The ROI from Software Quality." Auerbach Publications, Edition 1, ISBN-10: 9780849332982, ISBN-13: 978-0849332982, (June 2005).
7. Nair, Gopalakrishnan T.R., Suma V., Nair, Nithya G. "Estimation of Characteristics of a Software Team for Implementing Effective Inspection Process through Inspection Performance Metric". Software Quality Professional Journal, American Society for Quality (ASQ), Volume 13, Issue 2, (March 2011) 14-26.

Free and Open Source Software Use

Benefits and Compliance Obligations

Philip Koltun, The Linux Foundation

Abstract. Many systems developed for and deployed by the U.S. government now use Free and Open Source Software (FOSS). But FOSS use comes with potential license obligations. Essential compliance activities include identification of FOSS used in products along with communication of a FOSS bill of materials; review and approval of planned FOSS use; and satisfaction of license obligations. Compliance policies, processes, training, and tools enable contractors and government sponsors to use FOSS effectively. The Linux Foundation's Open Compliance Program provides many resources to assist with compliance.

Introduction

FOSS has been widely adopted as the software of choice in many core areas of computing. Linux dominates today in embedded systems and in servers, and other FOSS has gained widespread acceptance for operational use. Vibrant communities support Linux kernel development and many popular FOSS packages.

FOSS is all about freedom—freedom to use, study, modify, and distribute software under an open source license. Think of the word "free" as in free speech, not as in free beer.

The DoD has clarified[1] that FOSS use is acceptable and can provide significant benefits: high quality, reliable, and secure software resulting from continuous and broad peer-review; availability of source code for modification, which enables rapid response to changing situations, missions, and threats; avoidance of vendor lock-in; freedom to use and deploy in any context; low total cost of ownership; and so on.

Beyond the use of Linux, FOSS can be found in many domains, including (to name a few) software development tools and environments; computing infrastructure; graphics; mapping and geospatial imaging; modeling and simulation; communications and networking; security; database; and real-time computing. Indeed, a 2003 MITRE Corporation study[2] identified 115 FOSS applications in use in the DoD. Undoubtedly that number has grown quite substantially in the years since. The DoD has published an online Frequently Asked Questions (FAQ) page that dispenses useful information about DoD use of FOSS.[3]

With FOSS use comes responsibility. Typical license obligations consist of inclusion of attributions, copyright notices, and license text along with the product when it is distributed externally. Providing complete and corresponding source code or an offer of source code may also be required, depending on the FOSS licenses involved.[4]

Normally, license obligations are triggered when external distribution of a product occurs. The entity that distributes a product containing FOSS bears the responsibility for meeting relevant license obligations; they can not just point at an upstream supplier and say, "See them for whatever you are entitled to." On the other hand, what constitutes "external distribution" may be subject to legal interpretation. The aforementioned FAQ indicates that as long as a product acquired or developed by the U.S. government is not conveyed outside the U.S. government, external distribution has not occurred.[5] As a result, use of the software within the U.S. government context normally would not trigger license obligations.

Why, then, should the defense software community served by CrossTalk concern itself with FOSS compliance issues? At least two perspectives are worth examining. First is that of the DoD contractor delivering software to the government who uses FOSS to implement required functionality. Second is that of the government program manager overseeing the contractor and assuring that the government receives the freedoms, rights, and information to which it is entitled.

The contractor must assure that it knows what FOSS is included in its deliverable software and that it can satisfy any license obligations, so that the government will be able to enjoy its freedoms. Inasmuch as a contractor may use subcontractors, the task of knowing what is in the delivered code can be somewhat demanding.

The government, on its side, has an interest to preserve its options to distribute software to allies or to the public, actions that might trigger FOSS license obligations. So the government's interest is to assure that software delivered to it by contractors comes with all necessary freedoms. As a result, contractual agreements should require FOSS disclosure and FOSS obligation satisfaction from its suppliers. The government should also investigate its suppliers' FOSS compliance practices as part of its background diligence in contracting. Does a supplier have a policy on FOSS use, compliance training for its teams, automated code scanning to facilitate discovery and recognition of FOSS inclusion, a procedure to prepare a FOSS bill of materials, and so on? The Linux Foundation's "Self-Assessment Checklist" can be used effectively to assess supplier compliance practices and engage suppliers in discussion about compliance[6]. There would be good reason, as well, to incorporate FOSS compliance discussions in SEI assessments conducted to qualify the contractor.

FOSS Compliance

FOSS compliance refers to the aggregate of policies, processes, training, and tools that enables an organization to effectively use FOSS and contribute to open communities while respecting copyrights, complying with license obligations, and protecting the organization's intellectual property and that of its customers and suppliers.

What business processes enable organizations to comply with license obligations and project managers to assure obligations are satisfied? For a product being distributed externally, compliance involves three core activities: identification of FOSS; review and approval of planned use of FOSS; and satisfaction of license obligations for the included FOSS. Each of these will be discussed further below.

Identification of FOSS

First, identification of all FOSS in a product comes from the dual processes of disclosure and discovery. With disclosure, engineers and product managers of the contractor and its external suppliers typically identify FOSS based on prior knowledge of where the code came from. Discovery refers to audits (either manual or automated) that are used to identify FOSS code and its origin.

Reliance only on disclosure can be problematic. Few products these days are written from scratch. Most evolve from legacy products and externally acquired source code (either FOSS or commercially licensed software), with new code being written to implement differentiating features and functionality. Sometimes millions of lines of code may be included in a product, some of it pre-dating the engineers currently working for the company. It is unlikely that any one individual or team will know all of the code and where it came from. So it is hardly surprising that disclosure alone would be incomplete or inaccurate. Commercial scanning tools aid in the discovery process and are marketed by companies such as Black Duck Software, OpenLogic, Palamida, and Protecode, among others.[7] A number of open source scanning tools are also available.[8]

New technologies are being developed to codify and communicate in standard format a FOSS bill of materials. For instance, the Software Package Data Exchange specification (SPDX™), version 1.0, was released in the fall of 2011.[9]

Review and Approval

Reviewing and approving planned FOSS use is the second essential activity in compliance, typically requiring a panel of skilled and knowledgeable individuals known as an Open Source Review Board (OSRB). An OSRB must review FOSS use in context, so a product architectural diagram will be needed to show how the product's software components (including FOSS) interface and interact. The OSRB examines licensing implications of the architecture, compatibility of components from a license perspective, and resultant license obligations. Therefore, an OSRB must incorporate the expertise of skilled software architects and licensing experts.

Satisfaction of Obligations

The third essential activity in a compliance program concerns satisfaction of FOSS license obligations. Many organizational actions must come together to assure obligations can be met. As stated earlier, obligation fulfillment typically involves inclusion of attributions, copyright notices, and license text along with the product when it is distributed externally. Providing complete and corresponding source code or an offer of source code may also be required, depending on the FOSS licenses involved.

Ultimately, an effective compliance program must integrate compliance activities into day-to-day business processes so that identification, review and approval, and obligation satisfaction steps are routinely accomplished in time for scheduled product delivery.

Individuals or teams responsible for product documentation must perform necessary tasks to assure that documentation obligations are met.

As part of the process to satisfy source code obligations, the contractor typically should place into a software repository the complete source code corresponding exactly to each FOSS package used in a given product release. The complete source code may include any associated interface definition files, plus the scripts used to control compilation and installation of the executable. Verification activities should assure that source code used to produce product binaries has been cleansed of any inappropriate comments and that all FOSS packages in the product have been approved by the OSRB.

Ultimately, an effective compliance program must integrate compliance activities into day-to-day business processes so that identification, review and approval, and obligation satisfaction steps are routinely accomplished in time for scheduled product delivery. Key elements of a compliance program include company policy, employee training, assignment of compliance responsibility, staffing of the compliance function, and automation to enhance efficiency and accuracy. Compliance program implementation dovetails very nicely with CMMI®.[10]

Key process capabilities that must be brought to bear in compliance include supplier management, software configuration management, training, software architectural design and review, and verification, at a minimum.

Compliance Resources

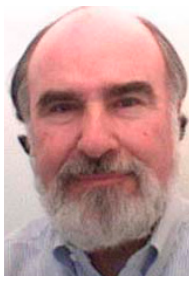
The Linux Foundation's Open Compliance Program is the software industry's only neutral, comprehensive software compliance initiative. By marshaling the resources of its members and leaders in the compliance community, the Linux Foundation brings together the individuals, companies, and legal entities needed to expand the use of FOSS while decreasing legal costs and reducing fear, uncertainty, and doubt.

Organizations seeking greater insight into compliance practices can take Linux Foundation compliance training courses; download freely available Linux Foundation compliance white papers and the Self-Assessment Checklist; participate in the SPDX working group; participate in the FOSSBazaar community and discuss compliance best practices; and access other helpful resources. More information can be found at <http://www.linuxfoundation.org/programs/legal/compliance>.

Disclaimer:

CMMI® is registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

ABOUT THE AUTHOR



Dr. Philip Koltun formerly directed the Linux Foundation's Open Compliance Program. He has implemented comprehensive open source compliance programs for Motorola and NAVTEQ, including policies and procedures, training, OSRB function, supplier compliance, and compliance tool introduction. Previously, he managed a software productivity and quality center at Harris Corporation; facilitated SEI, ISO, and Baldrige-style assessments; managed software suppliers; and consulted on business process improvement. He earned a Ph.D. in Computer Science from University of North Carolina-Chapel Hill.

The Linux Foundation
1796 18th Street, Suite C
San Francisco, CA 94107
Phone: (815) 353-2748
Fax: (415) 723-9709
E-mail: philip.koltun@gmail.com

NOTES

1. "Clarifying Guidance Regarding Open Source Software," October 16, 2009, from the DoD CIO, <http://cio-nii.defense.gov/sites/oss/2009OSS.pdf>
2. "Use of Free and Open Source Software (FOSS) in the U.S. Department of Defense," The MITRE Corporation, 2003, <http://cio-nii.defense.gov/sites/oss/2003survey/dodfoss_pdf.pdf>
3. See http://cio-nii.defense.gov/sites/oss/Open_Source_Software_(OSS)_FAQ.htm
4. The author is not a lawyer and this article should not be construed as providing legal advice. Please consult qualified counsel for interpretation of license terms and other questions requiring legal guidance.
5. See "DoD Open Source Software (OSS) FAQ" at <http://cio-nii.defense.gov/sites/oss/Open_Source_Software_(OSS)_FAQ.htm#Q:_Under_what_conditions_can_GPL-licensed_software_be_mixed_with_proprietary.2Fclassified_software.3F>
6. "Self-Assessment Checklist," The Linux Foundation, November, 2010, <http://www.linuxfoundation.org/programs/legal/compliance/self-assessment-checklist>
7. See <http://www.blackducksoftware.com>, <http://www.openlogic.com>, <http://www.palamida.com>, and <http://www.protecode.com>.
8. See, for example, the FOSSology tool at <http://fossology.org> and OpenLogic's Discovery tool, <http://www.openlogic.com/downloads/ossdiscovery.php>.
9. See <http://spdx.org>
10. <http://www.sei.cmu.edu/cmmi>

Congratulations to 2011 DoD Systems Engineering Top 5 Program Award Winners

SYSTEMS ENGINEERING GOVERNMENT AND INDUSTRY TEAMS

ARMY: **Army Integrated Air and Missile Defense (AIAMD)**
AIAMD Project Office / Northrop Grumman Corporation

ARMY: **Chinook CH-47F Multi-Year I**
Program Manager Cargo / The Boeing Company

NAVY: **Advanced Explosive Ordnance Disposal Robotic System (AEODRS)**
Naval Surface Warfare Center, Naval EOD Technology Division / Johns Hopkins University Applied Physics Laboratory (JHU/APL)

NAVY: **CH-53K Heavy Lift Replacement Helicopter (HLR)**
PMA-261 / Sikorsky Aircraft Corporation

AIR FORCE: **Enterprise Business Systems**
Air Force Research Laboratory / Jacobs Technology, Tybrin Group

http://www.acq.osd.mil/se

Deployment Optimization for Embedded Flight Avionics Systems

Brian Dougherty, Vanderbilt University
Douglas C. Schmidt, Vanderbilt University
Jules White, Virginia Tech
Russell Kegley, Lockheed Martin Aeronautics
Jonathan Preston, Lockheed Martin Aeronautics

Abstract. Loosely coupled publish/subscribe messaging systems facilitate optimized deployment of software applications to hardware processors. Intelligent algorithms can be used to refine system deployments to reduce system cost and resource requirements, such as memory and processor utilization. This article describes how we applied a computer assisted deployment optimization tool to reduce the required processors and network bandwidth consumption of a legacy flight avionics system.

Software Defense Application

The deployment topology of a distributed system determines how software is mapped to hardware. Optimizing the deployment topology of DoD distributed embedded systems has a significant impact on how efficiently the software utilizes the hardware. Deployment optimization can also help minimize costs by increasing hardware efficiency without requiring changes to the software or hardware architecture. This increase in hardware efficiency, in turn, helps reduce fuel consumption, increase operational ranges, and decrease cost.

Introduction

Current Trends and Challenges

Several trends are shaping the development of embedded flight avionics systems. First, there is a migration away from older federated computing architectures where each subsystem occupied a physically separate hardware component to integrated computing architectures where multiple software applications implementing different capabilities share a common set of computing platforms. Second, publish/subscribe based messaging systems are increasingly replacing the use of hard-coded cyclic executives.

These trends are yielding a number of benefits. For example, integrated computing architectures create an opportunity for system-wide optimization of deployment topologies, which map software components and their associated tasks to hardware processors as shown in Figure 1.[1]

Optimized deployment topologies can pack more software components onto the hardware, thereby optimizing system processor, memory, and I/O utilization [1, 2, 3]. Increasing hardware utilization can decrease the total hardware processors that are needed, lowering both implementation costs and maintenance complexity. Moreover, reducing the required hardware infrastructure has other positive side effects, such as reducing weight and power consumption.

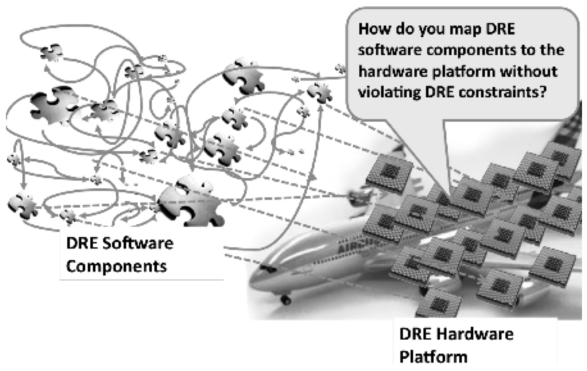

How do you map DRE software components to the hardware platform without violating DRE constraints?
DRE Software Components
DRE Hardware Platform

Figure 1. Flight Avionics Deployment Topology (© 2010 by Vanderbilt and Lockheed Martin)

Open Problems

Developing computer-assisted methods and tools to deploy software to hardware in embedded systems is hard [4, 5] due to the number and complexity of constraints that must be addressed.

For example, developers must ensure that each software component is provided with sufficient processing time to meet any real-time scheduling constraints [6]. Likewise, resource constraints (such as total available memory on each processor) must also be respected when mapping software components to hardware components [6, 7]. Moreover, assigning real-time tasks in multi-processor and/or single-processor machines is NP-Hard [8], which means that such a large number of potential deployments exist that it would take years to investigate all possible solutions.

Current algorithmic deployment techniques are largely based on heuristic bin-packing [8, 9, 10], which represents the software tasks as items that take up a set amount of space and hardware processors as bins that provide limited space. Bin-packing algorithms try to place all the items into as few bins as possible without exceeding the space provided by the bin in which they are placed. These algorithms use a heuristic, such as sorting the items based on size and placing them in the first bin they fit in, to reduce the number of solutions that are considered and to avoid exhaustive solution space exploration.

Conventional bin-packing deployment techniques take a one-dimensional view of deployment problems by just focusing on a single deployment concern at a time. Example concerns include resource constraints, scheduling constraints, or fault-tolerance constraints. In production flight avionics systems, however, deployments must meet combinations of these concerns simultaneously.

Solution Approach: Computer Assisted Deployment Optimization

This paper describes and validates a method and tool called ScatterD that we developed to perform computer-assisted deployment optimization for flight avionics systems. The ScatterD model-driven engineering [11] deployment tool implements the Scatter Deployment Algorithm, which combines heuristic bin-packing with optimization algorithms, such as genetic algorithms [12] or particle swarm optimization techniques [13] that use evolutionary or bird-flocking behavior to perform blackbox optimization. This article shows how flight avionics system developers have used ScatterD to automate the reduction of processors and network bandwidth in complex embedded system deployments.

The remainder of this article is organized as follows: Section 4 outlines a flight avionics deployment case study we use to motivate the challenges and solutions throughout the paper; Section 5 describes the challenges faced by developers when attempting to optimize a representative flight avionics deployment topology; Section 6 discusses the ScatterD tool for deployment optimization; Section 7 provides empirical results demonstrating the reductions in hardware footprint and network bandwidth consumption that ScatterD can produce; and Section 8 presents concluding remarks.

Section 4:
Modern Embedded Flight Avionics Systems: A Case Study

Over the past 20 years, flight avionics systems have become increasingly sophisticated. Modern aircraft now depend heavily on software executing atop a complex embedded network for higher-level capabilities, such as more sophisticated flight control and advanced mission computing functions. To accommodate the increased amount of software required, avionics systems have moved from older federated computing architectures to integrated computing architectures that combine multiple software applications together on a single computing platform containing many software components.

The class of flight avionics system targeted by our work is a networked parallel message-passing architecture containing many computing nodes. At the individual node level, ARINC 653-compliant time and space partitioning separates the software applications into sets with compatible safety and security requirements. Inside a given time partition, the applications run within a hard real-time deadline scheduler that executes the applications at a variety of harmonic periods.

The integrated computing architecture shown in Figure 2 has benefits and challenges. Key benefits include better optimization of hardware resources and increased flexibility, which result in a smaller hardware footprint, lower energy use, decreased weight, and enhanced ability to add new software to the aircraft without updating the hardware. The key challenge, however, is increased system integration complexity. In particular, while the homogeneity of processors gives system designers a great deal of freedom allocating software applications to computing nodes, optimizing this allocation involves simultaneously balancing multiple competing resource demands.

For example, even if the processor demands of a pair of applications would allow them to share a platform, their respective I/O loads may be such that worst-case arrival rates would saturate the network bandwidth flowing into a single node. This problem is complicated for single-core processors used in current integrated computing architectures. Moreover, this problem is being exacerbated with the adoption and fielding of multi-core processors, where competition for shared resources expands to include internal buses, cache memory contents, and memory access bandwidth. Artifacts complete with data describing the computational interactions and requirements of this system were provided by the Systems and Software Producibility Collaboration and Experimentation Environment (SPRUCE) web portal <http://www.sprucecommunity.org>. The SPRUCE web portal allows industry partners to create challenge problems complete with artifacts comprised of real data. These problems can then be paired with members of the research community that maximize the potential of discovering new, innovative solutions.

Section 5:
Deployment Optimization Challenges

This section describes the challenges facing developers when attempting to create a deployment topology for a flight avionics system. The discussion below assumes a networked parallel message-passing architecture (such as the one described in Section 4).

The goal is to minimize the number of processors and the total network bandwidth resulting from communication between software tasks.

5.1 Challenge 1: Satisfying Ratemonotonic Scheduling Constraints Efficiently

In real-time systems, such as the embedded flight avionics case study from Section 4, either fixed priority scheduling algorithms, such as ratemonotonic scheduling, or dynamic priority scheduling algorithms, such as earliest deadline-first, control the execution ordering of individual tasks on the processors. The deployment topology must ensure that the set of software components allocated to each processor can be scheduled and will not miss real-time deadlines. Finding a deployment topology for a series of software components that ensures the ability to schedule all tasks is called "multiprocessor scheduling" and is NP-Hard [8].

A variety of algorithms, such as bin-packing algorithm variations, have been created to solve the multiprocessor scheduling problem. A key limitation of applying these algorithms to optimize deployments is that bin-packing does not allow developers to specify which deployment characteristics to optimize. For example, bin-packing does not allow developers to specify an objective function based on the overall network bandwidth consumed by a deployment. We describe how ScatterD ensures scheduling constraints are met in Section 6.1 and allows for complex objective functions, such as network bandwidth reduction.

5.2 Challenge 2: Reducing the Complexity of Memory, Cost, and Other Resource Constraints

Processor execution time is not the only type of resource that must be managed while searching for a deployment topology. Hardware nodes often have other limited but critical resources, such as main memory or core cache, necessary for the set of software components it supports to function. Developers must ensure that the components deployed to a processor do not consume more resources than are present.

If each processor does not provide a sufficient amount of

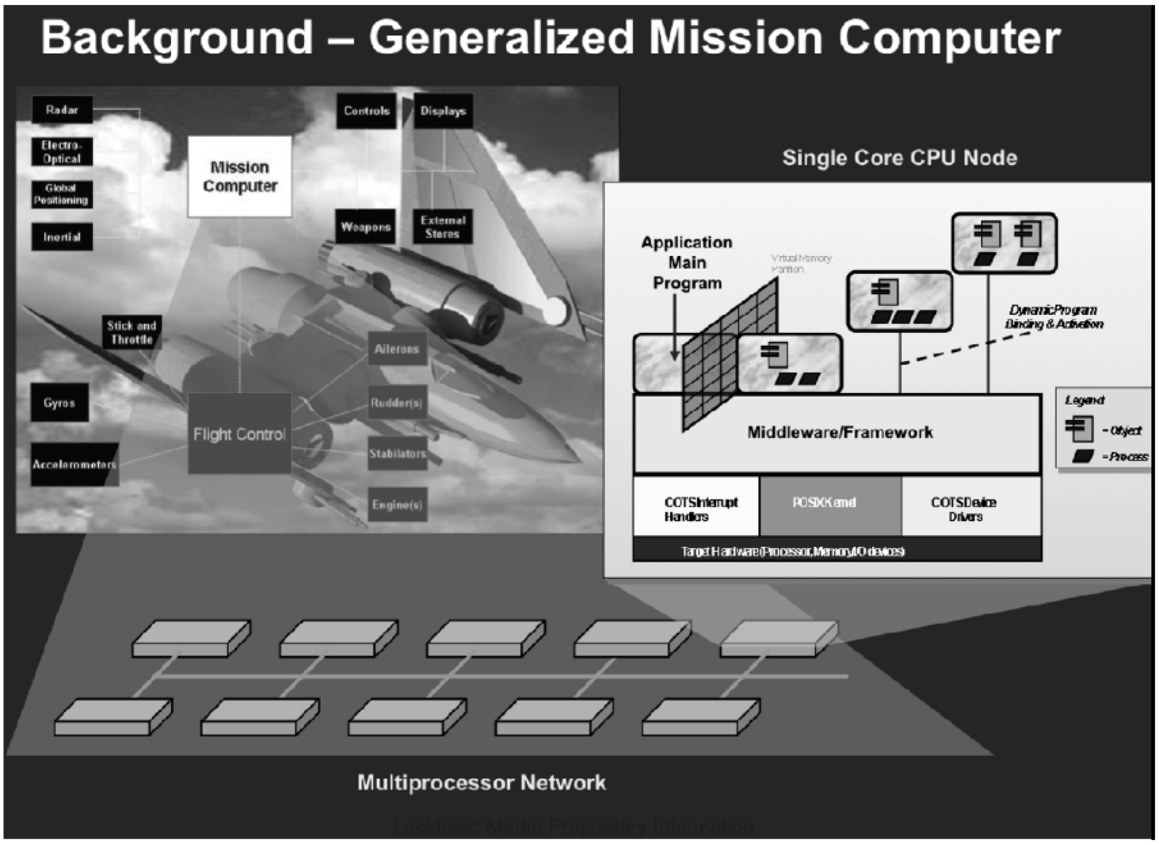


Figure 2. An Integrated Computing Architecture for Embedded Flight Avionics

resources to support all tasks on the processor, a task will not execute properly, resulting in a failure. Moreover, since each processor used by a deployment has a financial cost associated with it, developers may need to adhere to a global budget, as well as scheduling constraints. We describe how ScatterD ensures that resource constraints are satisfied in Section 6.2.

5.3 Challenge 3: Satisfying Complex Dynamic Network Resource and Topology Constraints

Embedded flight avionics systems must often ensure that not only processor resource limitations are adhered to, but also network resources (such as bandwidth) are not over consumed. The consumption of network resources is determined by the number of interconnected components that are not collocated on the same processor. For example, if two components are collocated on the same processor, they do not consume any network bandwidth.

Adding the consideration of network resources to deployment substantially increases the complexity of finding a software-to-hardware deployment topology mapping that meets requirements. The impact of the component's deployment on the network, however, cannot be calculated in isolation of the other components. The impact is determined by finding all other components that it communicates with, determining if they are collocated, and then calculating the bandwidth consumed by the interactions with those that are not collocated. We describe how ScatterD helps minimize the bandwidth required by a system deployment in Section 6.3.

Section 6:
ScatterD: A Deployment Optimization Tool to Minimize Bandwidth and Processor Resources

Heuristic bin-packing algorithms work well for multiprocessor scheduling and resource allocation. As discussed in Section 5, however, heuristic bin-packing is not effective for optimizing designs for certain system-wide properties, such as network bandwidth consumption, and hardware/software cost. Metaheuristic algorithms [12, 13] are a promising approach to optimize system-wide properties that are not easily optimized with conventional bin-packing algorithms. These types of algorithms evolve a set of potential designs over a series of iterations using techniques, such as simulated evolution or bird flocking. At the end of the iterations, the best solution(s) that evolved out from the group is output as the result.

Although metaheuristic algorithms are powerful, they have historically been hard to apply to large-scale production embedded systems since they typically perform poorly on problems that are highly constrained and have few correct solutions. Applying simulated evolution and bird-flocking behaviors for these types of problems tends to randomly mutate designs in ways that violate constraints. For example, using an evolutionary process to splice together two deployment topologies is likely to yield a new topology that is not real-time schedulable.

Below we explain how ScatterD integrates the ability of heuristic bin-packing algorithms to generate correct solutions to scheduling and resource constraints with the ability of metaheuristic algorithms to flexibly minimize network bandwidth and processor utilization and address the challenges in Section 5.

6.1 Satisfying Real-time Scheduling Constraints with ScatterD

ScatterD ensures that the numerous deployment constraints (such as the real-time scheduling constraints described in Challenge 1 from Section 5.1) are satisfied by using heuristic bin-packing to allocate software tasks to processors. Conventional bin-packing algorithms for multiprocessor scheduling are designed to take as input a series of items (e.g., tasks or software components), the set of resources consumed by each item (e.g., processor and memory), and the set of bins (e.g., processors) and their capacities. The algorithm outputs an assignment of items to bins (e.g., a mapping of software components to processors).

ScatterD ensures that all tasks of the flight avionics system discussed in Section 4 can be scheduled by using response-time analysis. The response time resulting from allocating a software task of the avionics system to a processor is analyzed to determine if a software component can be scheduled on a given processor before allocating its associated item to a bin. If the response time is fast enough to meet the real-time deadlines of the software task, the software task can be allocated to the processor.

6.2 Satisfying Resource Constraints with ScatterD

To ensure that other resource constraints (such as memory requirements described in Challenge 2 from Section 5.2) of each software task are met, we specify a capacity for each bin that is defined by the amount of each computational resource provided by the corresponding processor in the avionics hardware platform. Similarly, the resource demands of each avionics software task define the resource consumption of each item. Before an item can be placed in a bin, ScatterD verifies that the total consumption of each resource utilized by the corresponding avionics software component and software components already placed on the processor does not exceed the resources provided.

6.3 Minimizing Network Bandwidth and Processor Utilization with ScatterD

To address deployment optimization issues (such as those raised in Challenge 3 from Section 5.3), ScatterD uses heuristic bin-packing to ensure that all tasks can be scheduled and resource constraints are met. If the heuristics are not altered, bin-packing will always yield the same solution for a given set of software tasks and processors. The number of processors utilized and the network bandwidth requirements will therefore not change from one execution of the bin-packing algorithm to another. In a vast deployment solution space associated with a large-scale flight avionics system, however, there may be many other deployments that substantially reduce the number of processors and network bandwidth required, while also satisfying all design constraints.

To search for avionics deployment topologies with minimal processor and bandwidth requirements—while still ensuring that other design constraints are met—ScatterD uses metaheuristic algorithms to seed the bin-packing algorithm. In particular, metaheuristic algorithms are used to search the deployment space and select a subset of the avionics software tasks that must be packed prior to the rest of the software tasks. By forcing an altered bin-packing order, new deployments with different bandwidth and processor requirements are generated. Since bin-packing is still the driving force behind allocating software tasks, design constraints have a higher probability of being satisfied. By using metaheuristic algorithms to search the design space—and then using bin-packing to allocate software tasks to processors—ScatterD can generate deployments that meet all design constraints while also minimizing network bandwidth consumption and reducing the number of required processors in the avionics platform, as shown in Figure 3.

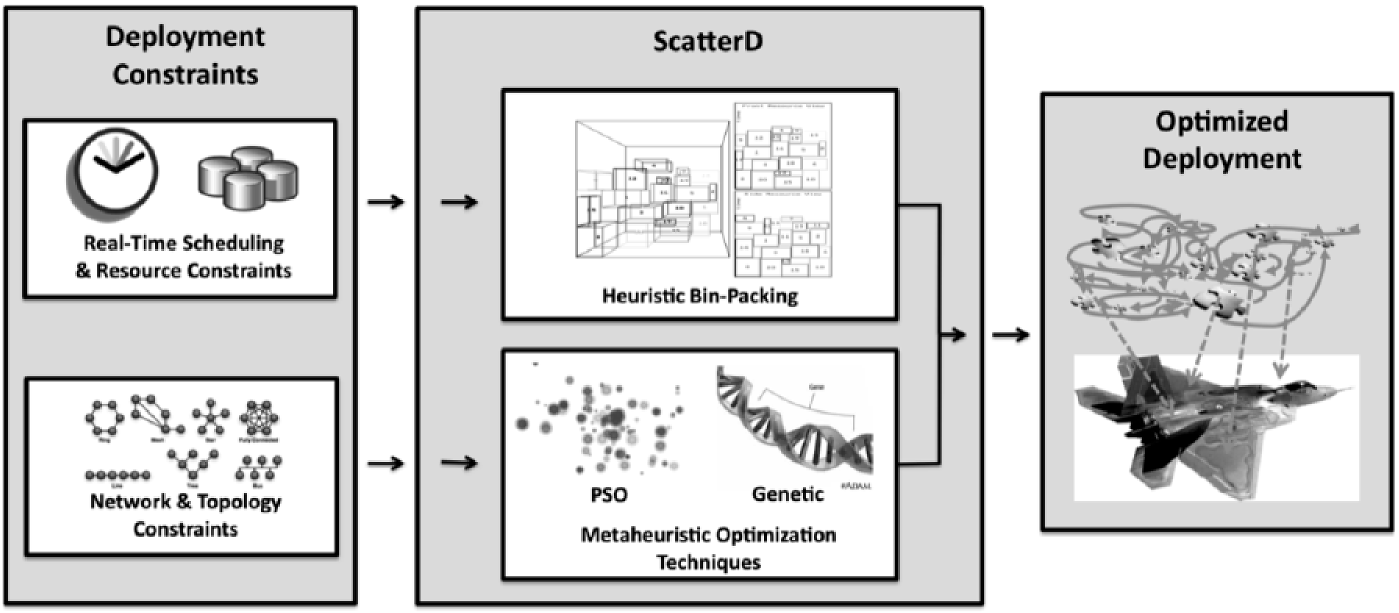

Deployment Constraints
Real-Time Scheduling & Resource Constraints
Network & Topology Constraints
ScatterD
Heuristic Bin-Packing
PSO
Genetic
Metaheuristic Optimization Techniques
Optimized Deployment

Figure 3. ScatterD Deployment Optimization Process

Section 7: Empirical Results

This section presents the results of configuring the ScatterD tool to combine two metaheuristic algorithms (particle swarm optimization and a genetic algorithm) with bin-packing to optimize the deployment of the embedded flight avionics system described in Section 4. We applied these techniques to determine if (1) a deployment exists that increases processor utilization to the extent that legacy processors could be removed; and (2) the overall network bandwidth requirements of the deployment were reduced due to collocating communicating software tasks on a common processor.

The first experiment examined applying ScatterD to minimize the number of processors in the legacy flight avionics system deployment described in Section 4. This system originally required 14 processors to support all necessary software tasks. Applying ScatterD with particle swarm optimization techniques and genetic algorithms resulted in increased utilization of the processors, reducing the number of processors needed to deploy the software of the system to eight in both cases. The remaining six processors could then be removed from the deployment without affecting system performance, resulting in the 42.8% reduction shown in Figure 4.

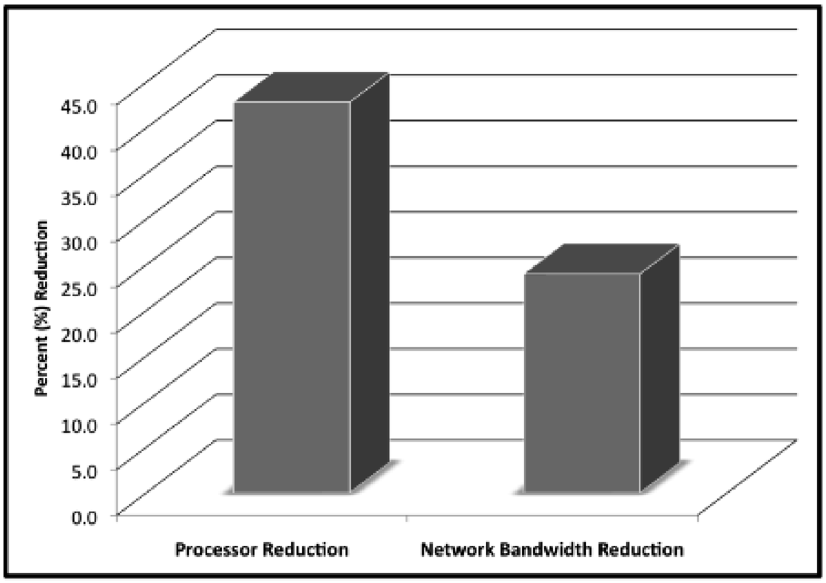


Figure 4. Network Bandwidth and Processor Reduction in Optimized Deployment

The ScatterD tool was also applied to minimize the bandwidth consumed due to communication by software tasks allocated to different processors in the legacy avionics system described in Section 4. Reducing the bandwidth requirements of the system leads to more efficient, faster communication while also reducing power consumption. The legacy deployment consumed $1.83 \cdot 10_{08}$ bytes of bandwidth. Both versions of the ScatterD tool yielded a deployment that reduced bandwidth by $4.39 \cdot 10_{07}$ bytes or 24%, as shown in Figure 4.

While these experiments prove the effectiveness of applying ScatterD to legacy system deployments, it is important to note that ScatterD can also yield benefits if applied when initially designing a new system. If the potential processor utilization and network interactions of the software tasks that comprise the system are known, then ScatterD can be applied to potentially yield a deployment with reduced processor requirements and network bandwidth consumption.

Section 8: Concluding Remarks

Optimizing deployment topologies on legacy embedded flight avionics systems can yield substantial benefits, such as reducing hardware costs and power consumption. The following are a summary of the lessons we learned applying our ScatterD tool for deployment optimization to a legacy flight avionics system:

• Multiple constraints make deployment planning hard. Avionics deployments must adhere to a wide range of strict constraints, such as resource, collocation, scheduling, and network bandwidth. Deployment optimization tools must account for all these constraints when determining a new deployment.

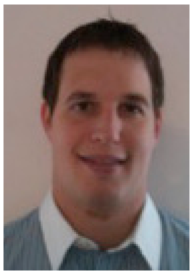
• A huge deployment space requires intelligent search techniques. The vast majority of potential deployments that could be created violate one or more design constraints. Intelligent and automated techniques, such as hybrid-heuristic bin-packing, should therefore be applied to discover valid "near-optimal" deployments.

• Substantial processor and network bandwidth reductions are possible. Applying hybrid-heuristic bin-packing to the flight avionics system resulted in a 42.8% processor reduction and a 24% bandwidth reduction. Our future work is applying hybrid-heuristic bin-packing to other embedded system deployment domains, such as automobiles, multi-core processors, and tactical smartphone applications.

• ScatterD can be applied throughout system lifetime. Systems may initially include expansion resources for inevitable system maintenance and to support new software that becomes available during the 20 to 30 year system lifetime. These expansion resources can be used to support new software that is added to the system overtime. Expansion resources, however, are finite and may not be necessary for a large portion of the system lifecycle leading to increased system weight and cost for an underutilized architecture. Therefore it is critical that all system resources, such as processor utilization and network bandwidth, are minimized so that superfluous hardware is limited. ScatterD can determine system deployments, and minimize network bandwidth consumption and processor utilization so that additional resources are present to support new software as it becomes available later in the system lifecycle.

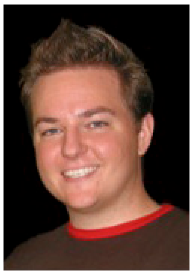
The ScatterD tool is available in open-source from the Ascent Design Studio <http://ascent-design-studio.googlecode.com>. A document describing the flight avionics system case study outlined in Section 4, as well as additional information on ScatterD, can be found at the SPRUCE web portal <http://www.spruceommunity.org>, which pairs open industry challenge problems with cutting-edge methods and tools from the research community.

ABOUT THE AUTHORS

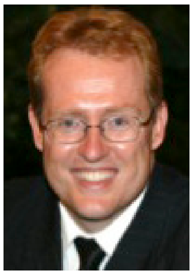


Brian Dougherty is a Ph.D. candidate at Vanderbilt University. Mr. Dougherty's research investigates automated techniques for configuring DRE systems and automatically scaling cloud computing applications to meet quality of service guarantees. He received his M.Sc. in Computer Science from Vanderbilt University in 2009.

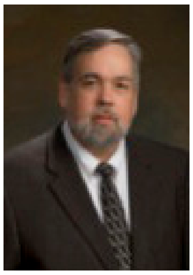
Brian Dougherty
Institute for Software Integrated Systems
Vanderbilt University
2015 Terrace Place
Nashville, TN 37203
E-mail: briand@dre.vanderbilt.edu



Jules White is an Assistant Professor in the Bradley Department of Electrical and Computer Engineering at Virginia Tech. He received his BA in Computer Science from Brown University, his MS and Ph.D. from Vanderbilt University. His research focuses on applying search-based optimization techniques to the configuration of distributed, real-time and embedded systems. In conjunction with Siemens AG, Lockheed Martin, IBM and others, he has developed scalable constraint and heuristic techniques for software deployment and configuration.



Dr. Douglas C. Schmidt is a Professor of Computer Science at Vanderbilt University. His research spans patterns, optimization techniques, and empirical analyses of software frameworks that facilitate the development of DRE middleware and applications. Dr. Schmidt has also led the development of ACE and TAO, which are open-source middleware frameworks that implement patterns and product-line architectures for high-performance DRE systems.



Russell Kegley is a Fellow at Lockheed Martin Aeronautics in Fort Worth, TX, where he works in the areas of real-time schedulability, software performance measurement and optimization, distributed algorithms, and internally-focused consulting. Some of his most rewarding experiences at Lockheed Martin are as a career mentor and C/C++ coach for younger engineers. He holds an MS in computer science from Mississippi State University.

Jonathan Preston is a Fellow at Lockheed Martin Aeronautics in Fort Worth, TX, where he works as a research lead and system/software architect. His interest areas include analytic methods, automated design and analysis techniques, and distributed real-time systems. He currently serves as a design consultant for multiple aircraft programs and is involved with several university and cross-corporate collaborations.

NOTES

1. This work was sponsored in part by the Air Force Research Laboratory under FA8750-08-C-0064 & FA8750-08-1-0025.

REFERENCES

1. L. Sha and J. Goodenough. Real-time scheduling theory and Ada. Computer, 23(4):53–62, 1990.
2. J. Strosnider and T. Marchok. Responsive, deterministic IEEE 802.5 token ring scheduling. Real-Time Systems, 1(2):133–158, 1989.
3. L. Lehoczky, J.P. snf Sha and J. Strosnider. Enhancing Aperiodic Responsiveness in a Hard Real-Time Environment. In Proc. of the IEEE Real-Time Systems Symposium, pages 416–423, 1987.
4. H. Beitollahi and G. Deconinck. Fault-Tolerant Partitioning Scheduling Algorithms in Real-Time Multiprocessor Systems. Pacific Rim International Symposium on Dependable Computing, IEEE, 0:296–304, 2006.
5. A.Carzaniga,A.Fuggetta,S.Richard,D.Heimbigner, A. van der Hoek, A. Wolf, and COLORADO STATE UNIV FORT COLLINS DEPT OF COMPUTER SCIENCE. A Characterization Framework for Software Deployment Technologies. Defense Technical Information Center, 1998.
6. J. Stankovic. Strategic Directions in Real-time and Embedded Systems. ACM Computing Surveys (CSUR), 28(4):751–763, 1996.
7. W.Damm,A.Votintseva,A.Metzner,B.Josko, T. Peikenkamp, and E. Bo¨de. Boosting Re-use of Embedded Automotive Applications Through Rich Components. Proceedings of Foundations of Interface Technologies, 2005, 2005.
8. A.Burchard,J.Liebeherr,Y.Oh,andS.Son.New Strategies for Assigning Real-time Tasks to Multiprocessor Systems. IEEE Transactions on Computers, 44(12):1429–1442, 1995.
9. S. Lauzac, R. Melhem, and D. Mosse. Comparison of Global and Partitioning Schemes for Scheduling Rate Monotonic Tasks on a Multiprocessor. In 10th Euromicro Workshop on Real Time Systems, pages 188–195, 1998.
10. Fault-Tolerant Rate-Monotonic First-Fit Scheduling in Hard-Real-Time Systems. IEEE Transactions On Parallel and Distributed Systems, pages 934–945, 1999.
11. D. C. Schmidt. Model-Driven Engineering. IEEE Computer, 39(2):25–31, 2006.
12. C. Fonseca, P. Fleming, et al. Genetic algorithms for multiobjective optimization: Formulation, discussion and generalization. In Proceedings of the fifth international conference on genetic algorithms, pages 416–423. Citeseer, 1993.
13. R. Poli, J. Kennedy, and T. Blackwell. Particle swarm optimization. Swarm Intelligence, 1(1):33–57, 2007.

FOR CONFERENCE & TRADE SHOW INFORMATION, VISIT WWW.SSTC-ONLINE.ORG
REGISTRATION OPENS JANUARY 2012

24th Annual
SSTC Systems & Software Technology Conference

SAVE THE DATE, PLAN NOW TO ATTEND

ACQUISITION
- Business & Information Systems
- Strategies, Policies and Standards

ARCHITECTURE
- Enterprise, Model-Driven, Open
- SOA

AGILE

CLOUD

CYBER PHYSICAL SYSTEMS
- NIST Standards
- Personal Cyber Defense
- Embedded Systems
- Run- time Operating Systems
- Integrated Systems
- Interoperability of Independent Systems

SECURITY/CYBER SECURITY
- Biometrics
- Identity Management
- Network Security
- Trusted
- Identity/Authentication
- Anti-Tamper
- Policy
- Information Assurance
- 20 Critical Controls

23-26 APRIL 2012
MARRIOTT DOWNTOWN HOTEL
SALT LAKE CITY, UTAH

WAR FIGHTING TECHNOLOGIES

ENHANCE ADVANCE MODERNIZE

Increasingly diverse and complex threats, coupled with decreasing budgets have provided the perfect incentive for the software and systems communities to increase attention to working smarter, being more innovative, and becoming much more efficient. The efficient and effective application of new and emerging technologies and methodologies is critical to helping warfighters, and those who support them, respond to those increasing threats. A revitalized SSTC 2012 will be an ideal forum for systems & software professionals to become smarter, gain a broader perspective of the choices and challenges they face, and collaborate with others who face similar challenges. Whether it's implementing cyber security, Better Buying Power acquisition guidance, or robust yet flexible architectures, or understanding how to apply agile development strategies or cloud computing solutions, SSTC 2012 will bring SW professionals together to discuss, learn, and collaborate. Plan now to embrace the challenge of becoming smarter, more innovative, and efficient. Join us at SSTC 2012 … then return to work better prepared to do more without more.

Upcoming Events

Visit <http://www.crosstalkonline.org/events> for an up-to-date list of events.

MILCOM 2011
7-10 November 2011
Baltimore, MD
<http://www.milcom.org>

GovSec West
14-16 November 2011
Phoenix, AZ
<http://govsecinfo.com/Home.aspx>

IEEE International Conference on Technologies for Homeland Security
15-17 November 2011
Waltham, MA
<http://www.ieee-hst.org>

Software Assurance Forum Working Group Sessions - Winter 2011
28 November - 2 December 2011
McLean, VA
<https://buildsecurityin.us-cert.gov/bsi/events.html>

Cybersecurity Conference
8-9 December 2011
Washington, DC
<http://foseinstitute.org/Home.aspx>

National Security Technology Expo
6-8 February 2012
San Diego, CA
<http://www.ubm.com>

Software Assurance Forum - Spring 2012
26-30 March 2012
McLean, VA
<https://buildsecurityin.us-cert.gov/bsi/events.html>

GovSec 2012
2-4 April 2012
Washington, DC
<http://govsecinfo.com/Home.aspx>

Systems and Software Technology Conference
23-26 April 2012
Salt Lake City, UT
<http://sstc-online.org>

Geek Mystique

The word geek is not new to the American vernacular. In fact, it has been around for a century or more, but its usage seems to be exploding. A quick web survey of websites found 15 sites using the term "geek" within their URL. Within the same search I found six major product lines using the word geek in the naming of their products. There are board games, movies, clothing lines, and categories of geeks ranging from physics and math geeks to sports and art geeks.

It is possible for a person to read Geek Magazine, Geek Weekly, or Fashion Geek—the Book. You can watch "Beauty and the Geek" on television, you can buy geek wear on hats, shirts, and coats. I have even seen the words "Geek 4 Life" tattooed on a person's arm, which seems to be a contradiction in lifestyles. Remember when it used to be the bad boy with the tattoo who picked on the poor, little, helpless, tattooless geek? The pop-culture acceptance of geekness has now blurred all those social categories. Face it, the term geek is far beyond cool and hip in the 21st century.

The word itself stems from early English and German dialects where it meant "fool." The root still survives in Dutch and German dialects today. The more Western European and eventual American definition was originally used to describe circus performers who performed disgusting acts and side show novelties like biting the head off a chicken.

We are all much more familiar with the modern definition which is "a computer expert or enthusiast." The term has been carried even further to now describe someone with super intellect and power. As we all know, it is now very chic to be a geek.

So how did the term geek outpace other terms and transform itself into the cool definition as it exists today? Why did other derogatory terms such as nerd and dweeb not make the same jump to social acceptance and then to status symbol? I will tell you why (and this is where I insert my opinion) but, it is an opinion with which I think you will agree. Geeks are cool because of the things they do which are exhibited everyday in software. They defend a nation. It can truthfully be said that the modern-day safety and defense of our nation rests, at least partially, on the shoulders of geeks. And that is one major reason why it is cool.

My personal opinion is our defense software engineering community is in personal possession of Super-Geeks; geeks that can do anything given the time to build it. It does not matter if they are computer scientists, electronic engineers, or some other derivative or hybrid of the two. They can do anything with software. I am completely and utterly impressed on a daily basis by the cool things our geeks do. The way they give new life to aging aircraft, the way they add new capabilities to previously listless systems, and the way they provide both physical and cyber security by the striking of a few keys on a keyboard.

I have entered the world of geekdom as a geek-geek, or in other words, I am geeky about geeks.

Just for fun, I began asking some of my co-workers if they considered themselves geeks, and if they welcomed the moniker. Here are two comments I received from some self-described geeks:

> "I have been called a geek and I felt it was a compliment."
>
> "I like being called a geek because geeks have style. I do not like the term nerd, because I think nerds have no social skills."

It appears there truly is a Geek Mystique and it is one of self-confidence, intellect, hipness, and to some degree—power. The geek shall inherit the earth, or from the looks of who is running things these days, they may already have.

Kasey Thompson
CrossTalk Advisor

Hill Air Force Base is hiring SOFTWARE ENGINEERS AND COMPUTER SCIENTISTS

Exciting and Stable Workloads:

- ★ Joint Mission Planning System
- ★ Battle Control System-Fixed
- ★ Satellite Technology
- ★ Expeditionary Fighting Vehicle
- ★ F-16, F-22, F-35
- ★ Ground Theater Air Control System
- ★ Human Engineering Development

Employee Benefits:

- ★ Health Care Packages
- ★ 10 Paid Holidays
- ★ Paid Sick Leave
- ★ Exercise Time
- ★ Career Coaching
- ★ Tuition Assistance
- ★ Retirement Savings Plans
- ★ Leadership Training

Location, Location, Location:

- ★ 25 minutes from Salt Lake City
- ★ Utah Jazz Basketball
- ★ Three Minor League Baseball Teams
- ★ One Hour from 12 Ski Resorts
- ★ Minutes from Hunting, Fishing, Water Skiing, ATV Trails, Hiking

Visit us at *www.309SMXG.hill.af.mil*. Send resumes to *shanae.headley@hill.af.mil*.
Also apply for our openings at *USAjobs.gov*

NAVAIR

CrossTalk thanks the above organizations for providing their support.